Simulative Performance Evaluation for the Design of Distributed Systems

Dissertation

Der Wirschaftswissenschaftlichen Fakultät der Universität Zürich

zur Erlangung der Würde eines Doktors der Informatik

vorgelegt von

Peter Lukas Weibel

von Gelterkinden BL und Schongau LU

genehmigt auf Antrag von

Prof. Dr. Lutz H. Richter
Dr. Reinhard Riedl

Dezember 2004

Die Wirtschaftswissenschaftliche Fakultät der Universität Zürich, Lehrbereich Informatik, gestattet hierdurch die Drucklegung der vorliegenden Dissertation, ohne damit zu den darin ausgesprochenen Anschauungen Stellung zu nehmen.

Zürich, den 8. Dezember 2004*

Der Lehrbereichsvorsteher: Prof. Dr. Martin Glinz

* Datum der Promotionsfeier

Abstract

Performance evaluations have mostly been measurements to determine the processing speed of a system or component. For the case of distributed systems the performance is often only tested when the system is used in either a test environment or even in the productive environment. It is only then that real usage scenarios, real amounts of data, and real effects of work load and disturbances are present and thus measurements realistic.

Many modern approaches allow the realization of all kinds of design conceptions for distributed systems. Only few of them seriously consider the performance aspect. In this thesis we present an approach that allows statements about usefulness and consequences of design conceptions for a system from the performance perspective even before the system has been realized or changed. The intention is a complement for systems design, not an examination after completion of a system's realization.

The core of our approach is an evaluation process that is closely integrated with the design process for a distributed system. The design model created there is translated into an evaluation model to be examined. The aim is to allow statements about resource usage, response time, and other performance indicators for the system's performance to find out whether the chosen system architecture can satisfy the requirements. Different usage scenarios can be used to do that.

Once an evaluation model is created, evaluation strategies are applied to gain knowledge about its performance. We present different strategies in this dissertation thesis. The so-called Cold Start Protocol, e.g., is a simple strategy to efficiently determine a throughput maximum for simple cases. More complex strategies have to be applied if the system usage is complex; they typically rely on the more simple strategies for their own realization.

The strategies are the core of our research. We use them to test hypotheses and to perform learning processes. They allow an evaluation system to execute standard tasks of performance evaluation without necessarily being controlled by an expert. A tool implementing these strategies is a means for designers to examine their design decisions by executing an evaluation, and even to compare alternatives directly. Even simple examinations of scalability are possible with this approach.

The strategies are realized using variation of specific parameters of the evaluation models. The variations refer to user-determined model

parameters. The strategies determine individual configurations for which a simulation experiment is executed. As a result of the simulation series, the strategies are able to determine the effects of the variation.

Finally, the results are presented in a suitable way, most as graphic representation. This representation in most cases contains the results of multiple experiments. It is aimed to facilitate the interpretation, and to support the users to draw the right conclusions from the evaluation.

Zusammenfassung

Performance-Evaluationen sind normalerweise Messungen, mit denen eine Verarbeitungsgeschwindigkeit eines Systems oder einer Komponente nachgemessen wird. Die Performance verteilter Systeme wird häufig erst untersucht, wenn die Systeme in einer Testumgebung oder sogar in der produktiven Umgebung eingesetzt werden. Erst dann sind die realen Benutzungsszenarien, Datenmengen, Belastungen und Störeffekte vorhanden, also die Messungen realistisch.

Viele moderne Ansätze erlauben die Realisierung aller möglichen Design-Konzepte für Verteilte Systeme. Nur wenige davon adressieren den Aspekt der Performance nachhaltig. In dieser Arbeit präsentieren wir einen Ansatz, der erlaubt, aus Sicht der Performance Aussagen über Nützlichkeit und Konsequenzen von Design-Konzepten im Gesamtzusammenhang zu machen, bevor ein System realisiert oder geändert wird. Es geht dabei um eine Ergänzung im System-Design und nicht um eine Überprüfung nach der Fertigstellung eines Systems.

Kernpunkt unseres Ansatzes ist ein Evaluationsprozess, der eng mit dem Designprozess für ein verteiltes System zu integrieren ist. Das Designmodell wird in ein Evaluationsmodell übernommen und dort untersucht. Es geht darum, vor der Realisierung Aussagen über die Ressourcennutzung, Antwortzeiten und andere Indikatoren für die Leistung zu machen und zu überprüfen, ob die gewählte Systemarchitektur den Anforderungen entspricht. Dabei können verschiedene Benutzungsszenarien zum Einsatz kommen.

Ist ein Evaluationsmodell erstellt, so kann mit verschiedenen Strategien Wissen darüber gewonnen werden. Wir stellen verschiedene Strategien in dieser Dissertation vor. Das sogenannte Cold Start Protokoll ist z.B. eine einfache Strategie zur effizienten Ermittlung des Durchsatzmaximums für einfache Fälle. Ist die Systembenutzung komplex, so kommen kompliziertere Strategien zur Anwendung, die jedoch zumeist auf die einfacheren Strategien zurückgreifen.

Die Strategien sind auch Kern unserer Forschung. Mit ihnen untersuchen wir Hypothesen und führen Lernprozesse durch. Sie erlauben einem Evaluationssystem, ohne permanente Steuerung durch einen Experten Standardaufgaben der Performanceevaluation durchzuführen. Damit wird Designern ein Mittel in die Hand gegeben, Designentscheidungen mittels Evaluation zu überprüfen und Alternativen direkt zu vergleichen. Das geht sogar bis zu einfachen Untersuchungen der Skalierbarkeit.

Realisiert werden die Strategien durch Variation, indem gewisse Parameter eines Modells durch die Strategien variiert werden können. Variationen beziehen sich auf durch Benutzer festgelegte Modellparameter. Die Strategien bestimmen einzelne Konfigurationen, für die dann jeweils ein Simulationsexperiment durchgeführt wird. Als Resultat der Simulationsreihe kann eine Strategie dann die Effekte der Variation ermitteln.

Schliesslich werden Resultate in geeigneter Form, zumeist graphisch, präsentiert. Diese Darstellung umfasst meist die Resultate vieler Experimente. Ihre Aufgabe ist, die Interpretation zu erleichtern und die Benutzer darin zu unterstützen, die richtigen Schlussfolgerungen aus der Evaluation zu ziehen.

Acknowledgments

My first and principal thanks go to Prof. Dr. Lutz Richter and Dr. Reinhard Riedl of the Department of Informatics (IfI) for accompanying and supervising my research and for having countless scientific discussions with me.

I would like to thank the Department of Informatics of the University of Zurich as well for making my work at the Department possible, and for providing the required infrastructure.

My thanks also go to the University of Dortmund, especially Prof. Dr. Heinz Beilner and Mr. Jürgen Mäter for allowing us to use HIT, the 'hierarchic evaluation tool' used for our implementation, and for repeatedly providing us several research licenses.

I thank Mrs. Carla Sadvary for proof-reading my text and making it sound a little less German and a little more English.

To my parents Gertrud und Dr. Heini Weibel many thanks for encouraging me to start this research project and for supporting me during that time.

Last but not least I would like to express my thanks and gratitude to my wife Charlotte, and my daughters Caroline and Madeleine for supporting me, taking me off the duties of daily life, having confidence in me, and at the same time enduring my so frequent absence from the family.

Contents

1 Introduction **1**

1.1 Motivation 2

1.1.1 The Situation 3

1.1.2 The Problem 5

1.1.3 Research Aims 7

1.2 The Approach 7

1.2.1 Simulation 8

1.2.2 Stochastic Elements of Modeling 8

1.2.3 Foundation of Performance Evaluation 9

1.2.4 Evaluation Process 10

1.2.5 Evaluation Instruments 10

1.2.6 Evaluation Strategies 11

1.3 The Remainder of this Thesis 11

2 Modeling for Evaluation **15**

2.1 Model Construction 16

2.1.1 Starting Point 16

2.1.2 Necessary Information 17

2.1.3 Building Evaluation Models 19

2.2 Workloads and Resources 29

2.2.1 Terms of Resources 29

2.2.2 Terms of Workloads 33

2.2.3 Evaluation Models 35

2.3 Foundations of Workload Modeling 37

2.3.1 Model Structure 37

2.3.2 A Priori Observables 41

2.3.3 Workload Elements as State Machines 43

2.3.4 A Posteriori Observables 45

2.4 Evaluation Results 49

2.4.1 Performance Maximum 49

2.4.2 The Throughput Graph 50

2.4.3 Comparisons and Effects 52

2.4.4 Scalability Comparison 54

2.5 Conclusion 54

3 The Evaluation Process **55**

3.1 Description of the Evaluation Process 56

3.1.1 Process steps and activities 56

3.1.2 Embedding the Process 62

3.1.3 Using the Process in Daily Life 64

3.2 Process Instruments 65

3.2.1 Instruments of Modeling 65

3.2.2 Evaluation Instruments 68

3.2.3 Instruments of Self-Adaption 69

3.2.4 Instruments of User Interaction 74

3.3 Evaluation Modeling Process Implications 75

3.3.1 Implications for Evaluation System Autonomy 75

3.3.2 Evaluation Model Evolution and Previous Experiments 76

4 Principles of Process Implementation **79**

4.1 Questions of Realization 80

4.1.1 Question Phrasing 80

4.1.2 Categories of Questions 81

4.1.3 Question Examples 84

4.2 Basic Implementation Concepts 86

4.2.1 Degrees of Freedom 86

4.2.2 Experiment Validity 87

4.2.3 Constraints 88

4.2.4 Capacity 90

4.3 Strategies 94

4.3.1 Strategies Description Systematics 94
4.3.2 Varying one Kind of Workload 96
4.3.3 Varying Multiple Kinds of Workload 98
4.3.4 Orders of Strategies 102
4.4 Strategies from the Users Perspective 103
4.4.1 User Requirements 103
4.4.2 Strategy Transparency 104

5 Process Implementation Strategies 107
5.1 Applicable Strategies 108
5.2 One-dimensional Localization Strategies 109
5.2.1 The Cold Start Protocol 110
5.2.2 Alternatives to Cold-Start 113
5.3 Localization Strategies for Multi-Dimensional Workload Variation 114
5.3.1 Basic Concepts 114
5.3.2 A Modified Cold-Start Protocol 115
5.3.3 The Warm-Start Protocol 115
5.4 Variations of the Model 115
5.4.1 Changing Resource Characteristics 116
5.4.2 Increasing Redundancy 119
5.4.3 Changing Model Structure 121
5.5 Scalability 123
5.5.1 Importance of Scalability 124
5.5.2 Basic Concepts 124
5.5.3 Constraints for Capacity Variation 124
5.5.4 Maximum for One Kind of Workload 125
5.5.5 Maximum for Multiple Kinds of Workload 126
5.5.6 The Simulation Quotient 126

6 Evaluation System Technology 129

6.1 The HIT Evaluation Tool 130

6.1.1 Characteristics of HIT 131

6.1.2 The Practical Use of HIT 135

6.2 Evaluation Framework in Perl 136

6.2.1 Experiment Execution 137

6.2.2 Implementation of the Strategies 138

6.2.3 Code Preparation 139

6.2.4 Preparation and Presentation of Results 141

6.2.5 Framework Structure 142

6.3 Tools and Utilities 145

6.4 Generic Architecture 147

6.4.1 Covering the Evaluation 147

6.4.2 Dynamics of the Process 148

6.4.3 Covering the Missing Parts 150

7 Results and Outlook 153

7.1 Overall Aims of this Thesis 153

7.2 Realizing the Aims 155

7.3 Specific Aims and Parts of the Evaluation 157

7.3.1 The Evaluation Process 158

7.3.2 Modeling 159

7.3.3 Simulation Techniques 160

7.3.4 Hypotheses 161

7.3.5 Strategies 162

7.4 Performance Evaluation Strategies 163
7.4.1 Hypotheses — Strategy Questions 163
7.4.2 Some Strategies Revisited 164
7.5 Research Results 165
7.5.1 Theory and Foundation 165
7.5.2 Process 166
7.5.3 Process Instruments 167
7.5.4 Variation and Strategies 169
7.5.5 Evaluation System Architecture 171
7.5.6 Presentation of Results 172
7.6 Potential for Further Research 173
7.6.1 Process 173
7.6.2 Interpretation of Results 175
7.6.3 Proximity of Experiments 176
7.6.4 Information Exchange 178
7.6.5 Human-Computer Interface 180

Appendices **183**

Terms of Performance Evaluation **185**

HIT Terms **191**

Bibliography **198**

List of Figures

2.1 A sample resource model depicting atomic resources on the lowest level, virtual resources (components) in the midth, and an insertion object generating workload elements on the highest level. 20

2.2 The vertical client / server principle, multi-level example . . . 21

2.3 File service example based on vertical client / server principle. 21

2.4 Example of horizontal client / server depicting web browser and web server. 22

2.5 Structural simplification of a resource usage hierarchy (left to right) 23

2.6 Simplification of behaviour (from top to bottom) 24

2.7 Example of the distribution principle depicting how the implementation of the service being addressed by the workload element implements the necessary communication effort. . . . 26

2.8 Example of the distribution principle depicting how the distribution is realized in an additional abstraction layer with a virtual resource that offers the remote service 'locally'. . . . 27

2.9 An example of a very simple resource hierarchy with atomic and virtual resources. 31

2.10 Generic Workload State Model 44

2.11 Example of an incomplete workload tree for a database update operation. The boxes on the higher levels depict abstract operations, i.e. services of virtual resources, those on the lowest level symbolize consumption of atomic resources. 47

2.12 Example of a graph depicting the throughput curve as function of the workload insertion rate. 51

2.13 Example of a graph depicting a 3-dimensional landscape of throughput maximization as function of the workload insertion rates. This example shows very few samples to show the effects of the Warm Start more clearly. 52

2.14 Troughput as function of the workload insertion rate. 53

3.1 Schema of the Evaluation Cycle. 58

3.2 Illustration of the different layers of evaluation. 62

4.1 Example of the effects of a constraint. The effort for coordination (on the right scale) grows with the number of nodes participating at the solution. The dotted ellipses depict combinations admitted by the constraint, other combinations are considered invalid. 91

4.2 Elements of a strategy description and their meaning. 95

4.3 Example of one-dimensional localization of the performance maximum. The small numbers for the points refer to the experiment number in the localization series. Notice that 26 out of 30 experiments in this example are for the approximation phase of the Cold Start. 99

4.4 Example of multi-dimensional localization of the performance maximum. The figure shows only three evaluation paths to keep it usable, a path meaning a ratio of different workload elements. The highest point of each path is the path's performance maximum. The performance maximum sought for is the set of these highest points. 102

5.1 Pseudo-code implementation of the first part of the Cold Start Protocol: Rough localization of the validity limit. 112

5.2 Pseudo-code implementation of the second part of the Cold Start Protocol: Narrowing the validity limit using an exponential bisection procedure. 113

5.3 Example of a scenario for scalability analysis. Multiple clients send requests in 'join-the-shortest-queue' policy to one of the application servers, which in turn sends its requests to the host. 127

6.1 The definition of resource properties in HIT using Hitgraphic's resource properties dialog. 131

6.2 Excerpt of one of HIT's generated result files. 135

6.3 Example of a result series compiled of information of HIT result files. 136

6.4 An example of an actual workflow showing its object-oriented hierarchic structure. The solid boxes are steps defined to start the workflow whild the dashed-line boxes are instanciated at run-time according to the actual situation. 138

6.5 Excerpt of a HI-SLANG code example before and after code modification (from 10 elements per second to 100 elements per second mean rate, bold typeface). The modification is very small in this case but it may include entire sections in other cases. 140

6.6 Left: Example of a data file made for GnuPlot; the data in the left column will be used as x-axis coordinates, the others as y-axis coordinates. Right: GnuPlot script to generate a corresponding graphics file. 143

6.7 An example of an actual workflow. The big boxes denote the steps with the workflow root on the top. The little boxes within show some of the steps' state. The arrows symbolize the flow of control, the bold arrows also mean instanciation. . 144

6.8 Schema of a value chain consisting of four member steps. The members are like categories, to which all parts of other process representations can be sorted into. 148

6.9 Schema of tools and parts identified earlier in this thesis and ordered according to the evaluation value chain. See figure 6.10 for the more complete schema. 148

6.10 Schema of the used tools now enhanced with the missing parts to be realized by a performance evaluation tool (Figure 6.9 completed). This illustration also includes the Evaluation Meta Data (EMD). 150

Chapter 1

Introduction

Performance is and has always been an issue for computer scientists. After all, much of today's work on computers is done because computer based work is cheaper and faster than manual or mechanical work. However, the performance considerations are often made in isolated context:

1. In the hardware industry, especially in the processor chip industry, there are broadly accepted benchmarks[1] serving as indicators[2] for a component's processing power.

2. In the area of system operations, there are indicators for 'delivered processing power' of a system or 'delivered bandwidth' of a network environment. These indicators are also quality measures for service level agreements and thus subject to permanent observation with systems management tools[3].

3. Finally, there are approaches of performance considerations originated in programming theory and algorithms. Both studies of efficient algorithms and complexity analysis are essential for the performance of modern systems.

[1] More on benchmarks can be found in [Eig01], [DG93], or [HPG02].

[2] Indicators are measures or statistic values allowing conclusions on the performance of an observed system. Cf. term 'key indicator' in section 7.6.5 for more.

[3] Cf. [oGC01] for more on service level management principles.

While we emphasize that all these considerations and approaches are necessary and useful we also think that they are not sufficient. Future software design and construction will more and more require the consideration of all the different performance aspects. This will make performance analyses necessary beginning in the early design phase of software construction.

In the past, experienced designers and programmers have been doublessly in the position to direct component design and construction into the right direction, although there was only little help on the tool side. In this world where more and more systems are getting connected in multiple ways, and mutual integration is widened on a daily basis, it appears to be important to investigate the performance aspect in a more integral and systematic way. We conclude that approaches and tools are required to study the performance better during systems design.

1.1 Motivation

Today, computers are so broadly available and seem so fast that analysis of system behaviour and performance may appear futile. However, there are many reasons to study it:

1. Software has to work with ever-growing amounts of data. Since storage hardware and computing power both grow, consumers demand more and more performance.

2. Performance indications for distributed systems are different from those of single-machine systems because distributed systems rely more on fundamentally asynchronous mechanisms and shared resources, e.g. the network. Although network communication is measurable and more or less predictable, it still remains subject to unexpected disturbances[4].

3. The growing number of dependencies makes reliability an important issue. For systems working with less reliable parts, reliability becomes a performance issue since reliability and recovery techniques have performance impact.

[4]Cf. [ZSN01] for some examinations of communication network phenomena.

4. Buying faster hardware makes a program run faster. While this may be true in the short run, performance improvements at the design level will certainly have more impact in the long run. They will make solutions faster even on fast hardware.

5. Systems are designed with a specific background of expected usage. Today, more and more systems and components are used much more and in a much broader way than originally planned for. The necessity of performance studies grows with the usage outside of the original scope.

   Ultimately, there may even be the wish for scalability analysis, i.e. the systematic study of performance improvements if a system is enhanced with additional components.

As mentioned in the introduction there is know-how about performance issues in both theory and practice. Therefore, it is important to rely on accepted theory and technologies. Based on these we propose an approach of performance evaluation integrated more into common software construction processes.

1.1.1 The Situation

Poor performance and little potential of improvement are often consequences of a way of designing systems that gives little room for holistic performance studies. We are convinced that there could be systems with better performance characteristics if analysis was taken into consideration early enough in the design phase. We believe that there is enough technical information for such considerations, either from product informations, or from measurements in an operative environment. However, there seem to be only few tools supporting the desired performance studies at the design phase.

Temporal Gap

Unfortunately, there is a gap between the time when the fundamental design decisions are taken and the time the system is operational and exposed to normal usage. By the time a system is really productive, with nearly the expected usage it was planned for, the design decisions with major influence

on the performance are already taken. In most cases, re-structuring the system at that time is too expensive and not desired for most solutions. The performance of the built system is a result of the ideas of the system architects and designers[5].

Model Gap

Performance models have often been isolated and very specific. There was little correlation with architecture or design models. System architects have designed systems without having a handsome tool to evaluate the performance of their design. Likewise, performance specialist have solved very specific problems that required their expert knowledge.

The way of modeling has often been so much different that it would have taken huge effort only to translate the architecture models into performance models. In addition, even if an excerpt of the system architecture was translated and tested the performance evaluation took place for an isolated part in an isolated context. It is in most cases delicate to draw conclusions for the design of a large system from the evaluation of an isolated part.

Understandability and Usability for Designers

Performance specialists have been rare in the past. In addition, they have often been focused on specific areas such as high-speed computing[6], clustering[7], or improving operations.

For designers on the other hand, performance is only one of the issues to be addressed during the design process. They often have performance questions for very specific issues of their design. In conventional performance evaluation approaches the solution has typically been to build an explicit model for that issue and to answer the question specifically. The interpretation of the results is often left to the performance expert, while the conclusions for the design are left to the designer.

Our approach is to bring these two specialists closer together technically. If there are tools and methodologies that are easier to understand by both designer and performance expert, the usage of a performance evaluation tool is encouraged and the collaboration between the two experts becomes closer.

[5]Some designers do text the performance of their components, but they often run the tests in an isolated way. Full integration tests are not always common practice.

[6]Cf [Wil88] or [Gin88] for more on high-speed computing and corresponding algorithms.

[7]Cf. [Ste01] or [Kop04] for examples of computer clustering.

Availability of Experts

As mentioned above, experts for performance evaluations are relatively rare. The lack of expert availability for broad system design has been one of the reasons why the methods of performance evaluations are not used more. However, not every performance evaluation activity requires an expert. There are routine activities that could be executed by designers if there were tools to collaborate with the commenced design process.

To improve the availability of some forms of performance evaluations for designers, there must be tool support to lead the users through the evaluation process and to present the results in a way that interpretation is easier for designers.

1.1.2 The Problem

From the current situation we can derive the problem to be addressed by our research and then design a plot of the target picture.

Temporal Issue

Since there is little helping us to conclude from a product's results to its early design we have to address the problem differently. Our idea is to use simulated results instead of real ones and to draw conclusions from simulations instead of real measurements.

Using an evaluation tool based on simulation allows us to study the system at design time, to make comparisons for design decisions, and even to make different studies for different usage assumptions.

Modeling Methodology

To make an approach convenient for designers and architects, the models must be made of building blocks very similar to those of systems modeling. Also, the interaction between those building blocks is also required to be similar since interaction is the analogy of processing.

On the other hand, there are conceptions of performance analysis and of design that need not to be reflected in the other model. For example, design modeling may contain many details of design not required for performance evaluation; vice versa the specific performance evaluation technique requires details not found in design models.

Understandability and Usability for Designers

Performance experts are not permanently available. Due to time pressure in projects, designers can often not afford to wait until there is an expert at their disposition. Performance evaluations must thus work with methodologies and techniques that can be understood and applied by designers. Modeling is certainly one of these areas, but there are others.

Performance evaluation must be implemented as a logical part of the design process. Design decisions will ultimately only be based on performance studies if the evaluation process can be integrated into and work with the steps of the design process.

Again, as a consequence, the building blocks of performance modeling must be made in a way that they can correspond with the design building blocks. For example, it is common to design in a systematic manner either top-down, or bottom-up, or inside-out. The performance evaluation process must have conceptions that allow support for this working manner, i.e. there must be support for services, interfaces, generalization, decomposition, refinement, and replacing structures. In addition, as it is common to model in levels or layers of abstraction, there must also be a corresponding conception for evaluation modeling.

Ultimately, there should also be support for embedded evaluations, i.e. the possibility to embed tests for specific components or structures into the whole performance evaluation model in a way that the embedded tests are executed 'for free'. This feature is a necessity, since it helps designers to confirm or reconsider their design decisions.

Availability of Experts

The availability of expert know-how is certainly required and justified for the improvements of system performance. However, there are many activities of performance studies not requiring the presence of an expert.

To increase the acceptance of performance evaluation as a design instrument, not only do the models and the process for performance evaluations have to be improved, but also the evaluation activities must be made available to system designers: There must be ways of executing routine activities of evaluating performance without experts. This should include the presentation of at least the simple results in a way that designers are able to interprete them and draw their conclusions.

1.1.3 Research Aims

The overall aim of this thesis is to provide a methodology and tools for engineers to learn about the performance of different architectures for distributed systems, and to be able to build systems with better performance characteristics.

To achieve that, the following objectives should be met:

1. It should be easy for designers to build models with artifacts which resemble the ones used every day.
2. It should be based on a well-known and accepted evaluation technology allowing to make results ready before the actual system can be tested in its target environment.
3. It should give designers means to compare different design alternatives with respect to their respective performance behaviour.
4. It should reach some level of autonomy in a way that designers can use the tools without permanent presence of a performance expert.

The elaboration will now be presented in the next section.

1.2 The Approach

The approach described in this thesis is based on several core concepts. Together they create a powerful and credible solution, address and solve many problems, and show even more potential for the success of performance evaluations in the future.

1.2.1 Simulation

We chose to base our performance evaluation technology on *discrete event simulation*[8]. It is a well-studied and accepted technology that allows us to gain information about the behaviour of a system before it can be tested in its target environment.

Discrete event simulation has the great advantage that its most famous programming language, Simula[9], is seen as the first object-oriented conception. Current design approaches[10] also use object-orientation as a fundamental principle. This makes similarity of design models and performance models much easier than for other approaches[11].

Simulation requires a formally defined model of building blocks[12] and interaction[13]. Since simulation tools come with their own way of defining and building simulation models, evaluation modeling should use these building blocks as much as possible.

Early in our research we have decided to use HIT™[14] as our modeling and experimentation tool. HIT comes with its own language, HI-SLANG, and with its own modeling tool, Hitgraphic™.

1.2.2 Stochastic Elements of Modeling

The use of discrete event simulation as evaluation technology allows us to express activities by means of stochastic expressions. Probability distributions are excellent means of expressing approximations of real-world situations with only little formal information.

Amongst the most important stochastic elements are the arrival processes defining, e.g., the arrival patterns of individual workload elements. We

[8] Cf. [PW93] for an introduction of discrete event simulation.

[9] Cf. `http://www.ifi.uio.no` or `http://www.simula.org` for more information on Simula and its roots at the university of Oslo, Norway.

[10] Cf. [JBR98a], [JBR98b], and [Mey00] for more on current design approaches

[11] With 'other approaches' we refer especially to analytical approaches or other forms of simulation.

[12] In object-orientation the building blocks are objects.

[13] Interaction is understood as the general term for communication, procedure calls, method calls, mutual usage, collaboration, and similar conceptions.

[14] HIT is a hierarchical performance evaluation tool developed by the University of Dortmund, Germany.

typically use a Poisson arrival process with the individual workload elements arriving in Poisson distributed intervals[15].

Other important forms of stochastic elements include probability based case decisions, e.g. to execute a specific branch of an `if` or `case` statement based on the value of a statistic variable. Uniform distribution is often required for this application.

As a consequence of using stochastic elements in modeling we have to consider that specific events do not happen deterministically at simulation time. If these events have major influence on the simulation results we are required to respect that at the definition of the simulation time. We typically do that by defining a simulation time long enough to be sure that the event will have happened often enough to be statistically relevant.

1.2.3 Foundation of Performance Evaluation

To build models, run experiments, and gather information consistently, a solid foundation is required. Such a foundation has to define the exact terms of the objects to work with, and it has to explain how the objects are defined, what properties they have, how they are grouped, and how they interact with each other. In addition, other conceptions of importance have to be introduced there.

Once model and object characteristics are defined and their mutual interaction is realized, we can have a look at the information that can be gathered for an experiment. While we know the workload types before execution, most of the information sought for is only known after an experiment. We are interested in statistical information about the model's ability to process workloads inserted from outside.

But we are also interested in reasons why a model reaches a specific desired or undesired state, why a workload element has to wait for processing, or why several workload elements do hinder each other. To analyze that, we will examine the structure of workload a posteriori.

[15] Cf. [Con98], [Hel98], or [PW93] for more on the Poisson distribution and its legitimation.

1.2.4 Evaluation Process

Learning about a system's performance is a process that typically takes far more time than individual experiments. The evaluation has to satisfy the following criteria:

- Facilitate testing for different scenarios
- Facilitate model changes due to progress in the design process; as a consequence enable re-executing previously executed tests
- Provide standard procedures for performance characteristics exploration
- Support model comparison

A process is required to both implement these tasks and realize the integration of all used instruments[16]. The process has to make sure that the activities are executed in correct order. After an interruption[17] it must be able to re-organize itself and resume the evaluation.

1.2.5 Evaluation Instruments

Instruments are specialized tools required to implement the evaluation process. The most important instrument is certainly the simulation instrument required to execute simulation experiments. Others include:

- Modeling instruments
- Evaluation series control instruments
- Instruments of Self-Adaption
- Result collection and presentation instruments
- Instruments of interaction with the users

[16] Cf. section 1.2.5 for more on the process instruments

[17] Interruptions should be part of the conception of an evaluation process, since the circumstances of evaluation may change at any time. E.g. a change of the design model requires a corresponding change of the evaluation model.

The instruments have different tasks:

1. Their first and primary task is the work they have been chosen or implemented for, i.e. simulation for the simulation instrument.
2. As a second task each instrument has to make sure that it can be called and controlled correctly. That means for external tools to issue the calls correctly and to monitor that they work correctly.
3. A final but important task is the correct usage of the context information. For experiment series that means that the simulation tool takes the parameters required for the specific simulation correctly.

1.2.6 Evaluation Strategies

Strategies implement the procedures of running experiment series. They are aimed at working according to procedures an expert might follow as routine part of his work. Different strategies are required for different tasks, e.g.:

- Localizing the performance maximum of a given model
- Comparing the performance behaviour of different models or of variations of a model
- Evaluating the scalability potential of a model

The strategies are thought of as understandable, comprehensive ways of addressing a common performance issue in an efficient way.

1.3 The Remainder of this Thesis

In chapter 2, we will introduce our way of modeling for performance evaluations. We will show the modeling activities, and we will cover the building blocks of performance modeling. We will define the precise terms of the objects and the conceptions required to understand and set up models. And, we will present a brief theoretical discussion on the objects,

their characteristics, and the information that can be gained by executing experiments.

Chapter 3 introduces and outlines the performance evaluation process. We explain the general understanding of the process and describe the process steps. The major process instruments are introduced and discussed. Afterwards, the implications of the process for modeling is looked at with respect to process execution autonomy and model evolution.

With chapter 4 we go into the depth of implementing the process. To do that we first have a look at the questions raised to implement a performance evaluation process. Then, we look at some concepts required for process implementation that have been identified during our experiment research. The major part of the chapter is devoted to the presentation of some strategies and their value. We close the chapter by illustrating the users' perception of strategies.

In chapter 5 we come to the realization of the performance evaluation process. The main focus of this chapter lies on the different strategies analyzed and applied during our research work. The first strategies realize localization techniques mainly used to find a performance maximum. The variations strategies deal with comparisons; they are mainly required to compare different models or different structures within a model but other differences can also be compared. To close the chapter, we present a brief discussion of an approach for scalability analysis showing that with our approach for performance analysis scalability can be investigated to a certain degree.

Chapter 6 gives an overview of the used base technology. The most important one is HIT, the performance evaluation tool made by the University of Dortmund, Germany. HIT comes with its own language, HI-SLANG, and its graphical modeling and experiment execution tool, Hitgraphic; we show the good features already offered by HIT, the parts that are missing, and ways how these parts could be implemented. Next to present is our framework programmed in Perl as workflow tool. We show how particular characteristics of the process can be implemented using our framework. We also take a quick look at other tools and utilities such as GnuPlot for graphic representations, e.g. To close this chapter we make a proposal how a good architecture for a performance evaluation tool should be built and organized; we show what parts can be filled with what tools, and we illustrate which parts have been realized as complements in Perl.

We conclude this thesis with chapter 7, where we start with a review of the research aims. We illustrate the different parts of the current solution and explain their contribution. Especially, we re-visit the described strategies. The review is concluded with a general overview over the achievements of our research. The remainder of chapter 7 is devoted to an outlook on the possibilities and opportunities of developing our approach of performance evaluations further and making it even more useful and credible for the users.

Chapter 2

Modeling for Evaluation

To study phenomena of performance both, theoretically and practically, we work with models representing the characteristics of the real world relevant for performance studies. We have to do that for two reasons: first, because the models have to be simple enough for users to build and to technically represent; and second, because relevant facts or findings must be clearly recognizable. This holds even more true for computer tool based modeling.

In the first part of this chapter we look at the aspects of the real world that are to be represented in such an evaluation model. We begin with a possible starting situation and the information needed as a base for building a model. Then we show how an evaluation model will be built in practical situations, to reflect the typical work procedures of system designers and other stake holders in a project.

To show how specific characteristics of the real world can be transformed into model representations, we will discuss a series of patterns of logic system design.

In the second part of the chapter we will introduce the building blocks of modeling, especially resources and workload. We will show their relevance for the execution of performance evaluations.

To illuminate the very details we will introduce the formal foundations for workload, resources, services, and events, and at the same time the duality of workload and services in the third part of this chapter. Doing so, we can illustrate characteristics of workload in an a priori and an a posteriori observation.

The fourth part is to show the transition of the evaluation model during the course of modeling and evaluation activities.

- We will show how to start with a simple model with coarse modeling simplifications, and how those simplifications are to be represented in an evaluation model. Also included are brief discussions of some stochastic phenomena that can be modeled with means of probability distributions.

- We will especially look at the decomposition of coarsely modeled services and the specification of real-world facts in the evaluation model.

- Finally, we show how the constructs of programming with significant influence on performance can be adequately represented in the evaluation model.

2.1 Model Construction

For the study of how to build a model we depart from a situation that might happen to system architects in their daily life. We will discuss what information is required to be able to set up an evaluation model, and we present ways to build new models.

2.1.1 Starting Point

Our approach to performance evaluation is aimed at evaluating constructions and models that do not yet exist, or that represent a major change to an existing system, respectively. We depart from the design of system architecture in different possible stages of development:

- In the best case, the evaluation model is built in parallel with the logical[1] and the physical[2] design. In this case, the information gained by using performance evaluation can be immediately used for the physical distribution design. Designers therefore prefer a model that can easily be changed, thus a model that is flexible with respect to changeability.

- Another plausible case is the construction of an evaluation model after having finished the logical and physical design. In this case, the evaluation is used as a means of validation[3]. The precise representation of interaction models[4] and distribution characteristics is mandatory for this case. The evaluation model should resemble as much as possible the logical and physical design model to allow for a appropriate interpretation of the evaluation results.

- In third case, an evaluation model is used to represent an already existing system to allow for systematically investigated system redesigns or new re-implementations[5].

Generally speaking there have to be some kind of concept for the system to be built or re-structured, its services, and the idea of distribution and physical deployment for any evaluation case.

2.1.2 Necessary Information

Essentially, 4 groups of information are necessary for the setup of an evaluation model.

[1] We will use the term 'logical design' as the design of an entire IT system that represents its functional elements and their relationships. A part of that is either a service hierarchy — a graph depicting that an A uses a B, which uses a C — or a component model similar to a UML package model. Please refer to [JBR98b] or [JBR98a] for more information on UML modeling.

[2] The term 'physical design' is used widespreadly and for many activities of computer science. For reasons of simplicity, we refer to physical design as the referral of logical components to physical resources, often called nodes. 'Best practices' approaches for IT operations often refer to physical design as 'deployment model' (cf. [oGC02] for the ITIL Standard in Application Management, or [JBR98b] the UML Notation for software design).

[3] Validation refers to checking the plausibility of component collaboration in this contex.

[4] Cf. UML Interaction Models in [JBR98b] or [JBR98a] for details.

[5] Cf. [ABB+02] for another study of capacity and performance analyses for distributed systems.

1. A hierarchy of services
2. A distribution concept
3. Information about the performance characteristics of all services
4. Information about the expected user or usage workload

Service Hierarchy

From a general perspective, every application execution consists of service use cases that refer — directly or indirectly — to resources of computers. Examples are:

- A calculation algorithm essentially uses CPU power and some memory.
- A database operation typically uses a lot of harddisk power, some CPU power and some memory.

We refer to this usage as **resource consumption**[6].

Virtual Resources After modeling some fundamental forms of resource consumption, we prefer not to program complex patterns of workload attribution for those resources in detail. Instead, we would like to model more abstract services with simple interfaces, which in turn refer to fundamental forms of resource consumption. In other words, our modeling approach should offer some form of abstraction. These abstractions are realized in model components that can be thought of as **virtual resources**.

As an example, we would like to model a service called 'database read operation', which is easy to use and in turn refers to one or more other resources to reflect appropriate runtime behaviour of a database system for its read operation. The service's implementation may consider the load and locking situation down to the database page level, the general load situation of the database's disk and CPU, abort and rollback situations for database transactions, and more. Or it could only reflect a lookup and fetch operation in a simple case.

[6] Cf. definition 8, section 2.2.1.

What behaviour is taken as model implementation depends on the scope of analysis and on the required preciseness. An application that has to run 5 database read operations during its execution will then use the service described above 5× instead of modeling the complex behaviour itself. In our terms the component offers an **abstract service** and plays the role of a **virtual resource** for that application.

Building Hierarchies The building principle denoted above can be used to build components, which – in turn – use other virtual resources. In modeling theory we call that a further 'level' or 'layer' of abstraction. Here too, one or more services are provided that refer to lower level services for realization. Their implementations create load for services of lower level resources, irrespectible whether those resources are virtual or atomic ones (Cf figure 2.1).

If we apply the just mentioned principle repeatedly, we will finally have a chain or hierarchy of components, which offer services on different levels of abstraction and all refer to services of atomic resources, either directly or transitively. The hierarchy is a directed graph with the resources depicted as nodes and the service usage at those resources as directed edges. Here, the sources of the graph are the resources with the most abstract services, and the sinks are the atomic resources, which are not decomposed any further.

2.1.3 Building Evaluation Models

With the construction approaches introduced above we are able to build hierarchically structured models which eventually represent the different levels of abstraction appropriately. Departing from different principles we will now illuminate model building in practice and defer some consequences for representation. The principles are:

- Client/server principle
- Principle of simplification
- Distribution principle
- Principle of the choice of appropriate physical resources
- Principle of proceeding at modeling

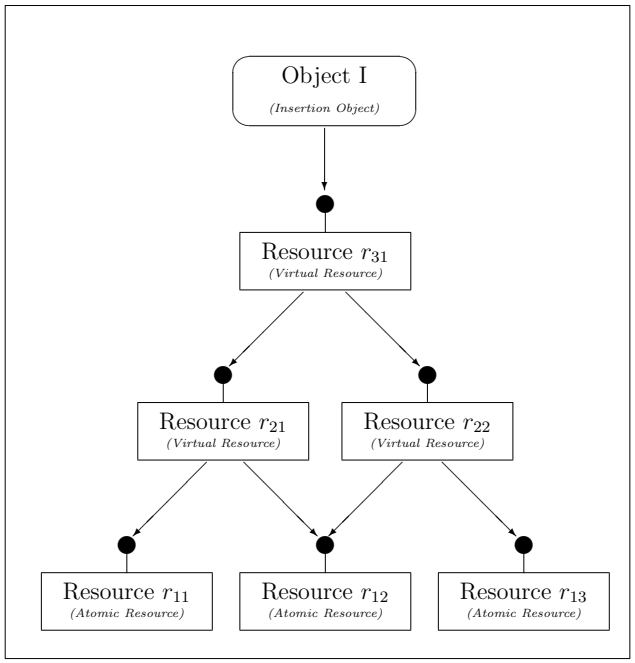

Figure 2.1: A sample resource model depicting atomic resources on the lowest level, virtual resources (components) in the midth, and an insertion object generating workload elements on the highest level.

Client/Server Principle

The Client / Server model[7] says:

- There are server elements that offer services and process service requests. The services are offered at an interface and can be used by all elements able to address the interface.
- There are client elements that use the servers' services by addressing the interface.

To facilitate usage of this model, we make a difference between the vertical and the horizontal client / server principle[8]. **Vertical client / server**, as depicted in Figure 2.2, refers to a layering model, where some functionality at higher levels uses some other functionality at lower levels.

[7]For an in-depth introduction to 'client/server computing' please refer to [OH98].

[8]The terms 'horizontal' and 'vertical' client / server principle were used here for additionally illustrating the (physical) difference between these logically similar approaches.

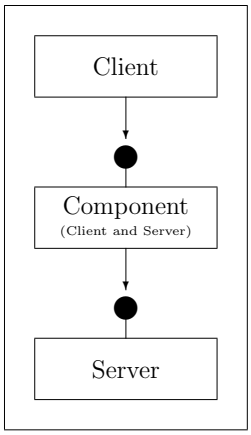

Figure 2.2: The vertical client / server principle, multi-level example

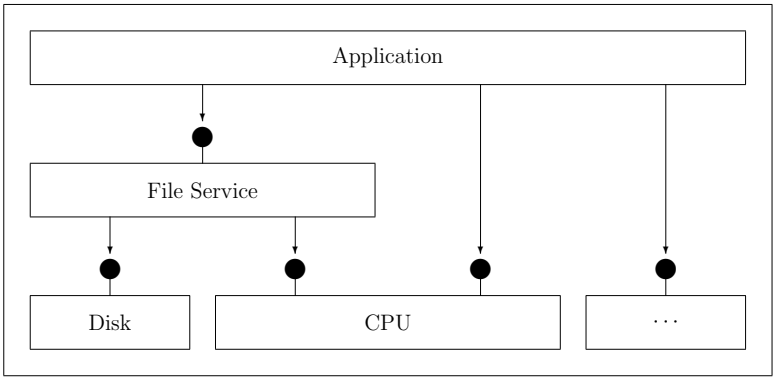

Figure 2.3: File service example based on vertical client / server principle.

The used functionality must be addressable at the corresponding component's interface and made available as a service. This principle can be found in almost all our models. As depicted in figure 2.3, a logical representation of a file service can be expressed by means of the vertical client / server principle.

However, if we bring physical aspects into consideration, another kind of client / server called the **horizontal client / server** principle is needed. Using this principle, we can model a component A that uses a service of a component B, which is not at the same place or within the same context.

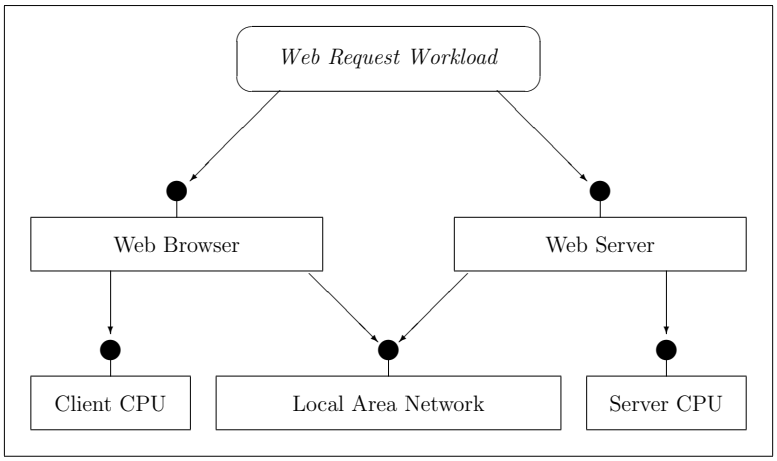

Figure 2.4: Example of horizontal client / server depicting web browser and web server.

Figure 2.4 shows an example of a web browser application that requests web pages from a web server[9]. Logically, the web server is a component delivering its service over an interface. When it comes to representing location or accessibility aspects, that vertical connection no longer holds true. Rather, a connection is built by means of a shared resource used by both, typically a communication component.

To model an appropriate sequence of activities, a workload type must be defined that consumes the client's and the server's services in appropriate order. E.g., for a single web request with only one concerned document the workload must issue a request to the client, then one to the server, and finally, one to the client again.

In our example, both client and server generate workload for the network component. If the network shows phenomena of heavy load, those phenomena mostly have impact on the whole model's performance characteristics.

Using this technique, we are able to express usage relationships between any kind of components that are linked somehow but not in the vertical client / server way.

[9]This can be roughly seen as a stateless file service.

Principle of simplification

We have already explained that an evaluation is based on a model representing a simplified view of reality. Simplification is a means for modelers to express the concept of some system within reasonable time and without going too much into detail. It is a legitimate means of representing an excerpt of a system allowing to understand, observe, or measure some phenomena.

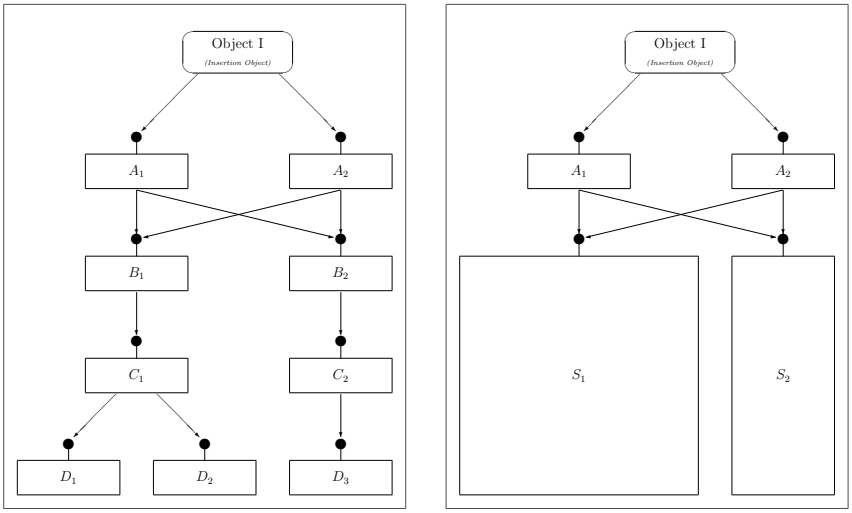

Figure 2.5: Structural simplification of a resource usage hierarchy (left to right)

During our studies, we have worked with two kinds of simplifications: one in structure and another one in behaviour. **Structural simplification** reduces the number of structural elements by using a black box[10] that shows the desired behaviour instead of a complex structure of elements.

Figure 2.5 shows two resource usage hierarchies to be replaced in a way that the entire hierarchy is represented in one simplified resource. Level A remains, because of its broad call structure, levels B through D disappear and get replaced by S_1 and S_2 resources.

The complex structure of an application's network access may serve as an example: Instead of the program, the memory control, the cache control, the main memory bus, the device drivers, user-level and system-level execution code, the CPU, the network protocol stack, the network card and finally

[10] Cf. [Bei95] for more on the principles of black-boxing.

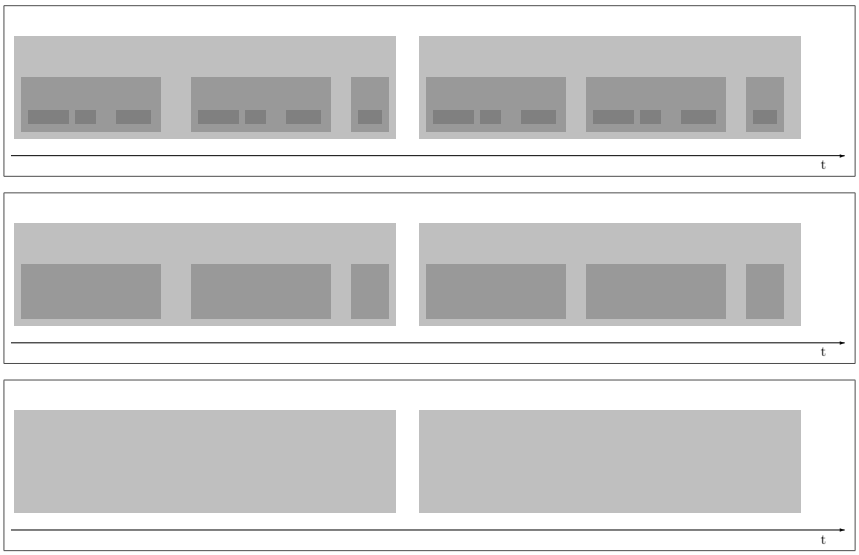

Figure 2.6: Simplification of behaviour (from top to bottom)

the network, a simplified resource 'network' or 'network access' may serve as appropriate enough representation for the model.

Although we meant 'simplification by the modelers' in this section, our approach uses this simplification principle exhaustively by offering an aggregation mechanism that enables performance engineers to capture the behaviour of an entire sub-structure and replace it by one aggregate resource[11].

Behavioural simplification is used by modelers to reduce the complexity of implementation. Modelers depart from a situation, where the processing of one workload element internally creates other workload elements, which in turn create more internal workload and so on. Instead of implementing a service by creating lower level workload, modelers could also choose to imlement within one resource in a way that its run time behaviour shows the same characteristics[12].

Figure 2.6 depicts a simplification of behaviour from very detailed behavioural patterns (in the top box) down to a very simple pattern (in the bottom box).

[11] Cf. section 3.2.3 for details on aggregation.

[12] With run-time characteristics we essentially refer to the time that is consumed and / or the internal load that is created.

Behavioural simplification is clearly in close relationship with structural simplification. We found that structural simplification is often motivated by considerations about behavioural simplification. There seems to be little to motivate structural simplifications other than behavioural analysis.

As a consequence of simplification, speed increases while preciseness decreases. With speed we refer to both, time to build a model, and time to evaluate a model[13]. While evaluations may require less time, the represented facts are clearly less. As a result, there is a loss in evaluation result data.

Whether the 'lost' data is relevant or not — i.e. whether the data are relevant information or not — must to some degree be left to the users. Experts typically start with very simple models and then add more detail for elements they identify to have key roles from past simulations.

Principles of Distribution

Every resource pool is situated at a specific location[14]. A location can either be a geographic or logic location, or it can also be an extent[15]. The modeling principle for locations says that services can only be addressed and used locally.

Let us look at an example with a client computer A and a database server B, B offering database services such as read or write operations, and transactions. The difficulty arising from the location modeling principle, is that a program on A cannot access the services on B. To make the access possible, there must be some connection that makes the services of B local to A. This connection is subject to the 'distribution principle'.

The modelers have 2 possibilities, to encode a remote access, both based on the same principle:

1. Realization within the abstract workload implementation of the caller: In this case, the calling service makes use of local, remote, and connecting services. For example, a typical usage pattern is: a local work portion, a communication, a remote work portion, another communication, and another local work portion. The fact that a request for

[13]Especially simulations run markably faster depending on the extent of simplification.
[14]Cf. 2.3.1 for more specific information about the term location.
[15]Consider the example of a communication network. It offers local services to every connected computer.

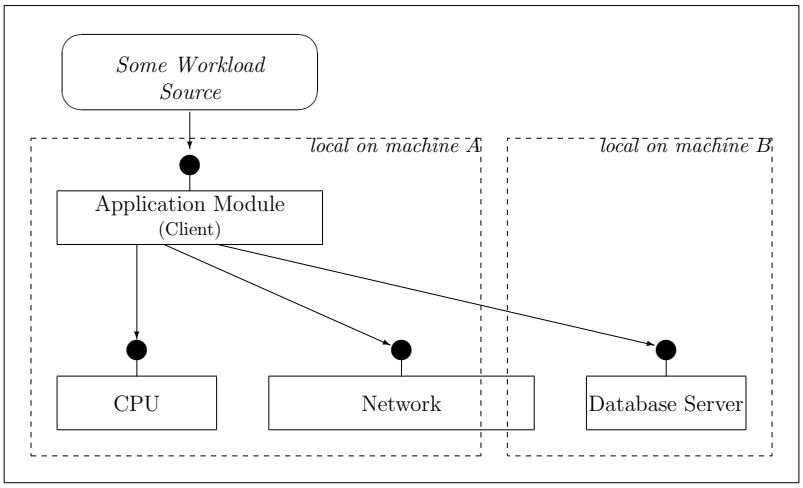

Figure 2.7: Example of the distribution principle depicting how the implementation of the service being addressed by the workload element implements the necessary communication effort.

a remote service requires communication effort is represented in the program code of the calling service. This example is shown in figure 2.7.

2. Realization by means of a virtual local resource: As an alternative, modelers can choose to introduce a virtual resource that offers its service locally and implements the correct sequence of accesses and communication in its service program code. The distribution is modeled the same way but on a lower level of abstraction. This approach typically has negative impact on the efficiency of simulation, since processing it needs more simulation time. However, it may be appropriate and useful to model an abstraction in a performance model the same way as it is represented in other models[16]. An example is shown in figure 2.8.

Although being less efficient in simulation, abstraction is often a key approach to distributed computing. E.g., to make remote services look local is a key concept in RPC, CORBA, or Java RMI[17].

[16]The representation in a way closely related to other modeling approaches for architecture or design models, is seen as an important reason for the acceptance of performance evaluations using our approach.

[17]Cf. [BN84] or [Sal96] for RPC, [OH98] or [Gro02] on CORBA, and [MvNV+99] or [Sun97] for more on RMI.

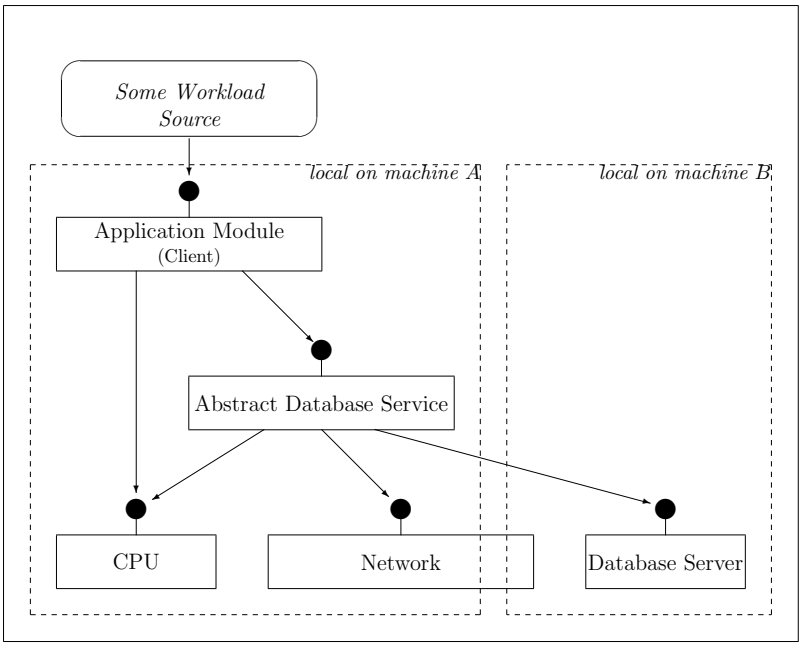

Figure 2.8: Example of the distribution principle depicting how the distribution is realized in an additional abstraction layer with a virtual resource that offers the remote service 'locally'.

The Choice of Appropriate Physical Resources

When it comes to modeling atomic physical resources, a natural choice is always the selection of service delivering devices. If studying a little local system in detail, the devices might be, e.g.:

- the CPU
- the main memory
- the secondary memory, typically a hard disk

For even more precise investigations, some of the following elements may be subject to modeling:

- the memory bus between CPU and main memory
- the data bus as interface between motherboard and disk

- primary and secondary cache

The reader has noticed already that the modelers have to define the granularity of representing atomic resources themselves. If investigating a small distributed system, the atomic resources to be modeled in the first iteration would probably be:

- the CPU of n clients
- the CPU of m servers
- the network

The choice of atomic resources has essential consequences, but can be modified during the course of modeling and experiments. If that happens, users may have to appropriately change all service implementations, which use or have used that particular resource.

Principle of proceeding at modeling

Hierarchic modeling exhibits remarkable advantages for the course of model building. During our research activities, we have found the following three principles of model or design composition to be useful for evaluation modeling:

- modeling *Top-Down*[18]
- modeling *Bottom-Up*[19]
- and modeling *Inside-Out*[20].

[18]Top-Down meaning that users start modeling at the highest level of abstraction and decomposing services down to physical resource usage.

[19]Bottom-Up meaning that users start modeling at the physical resource level continously building more and more abstract services, until eventually the services to be represented are defined.

[20]Inside-Out modeling meaning that users start at well-known services, arbitrarily moving up and down in abstraction, until eventually a service hierarchy is fully defined as in the upper cases. All three, Top-Down, Bottom-Up, and Inside-Out are design principles that have been applied for a long time in all engineering disciplines. Please refer to [Som92] for an in-depth discussion of these principles.

Our observations have shown that users often start with a bottom-up approach for new problems. If a model of logical or physical design exists already, we did rather find top-down proceeding[21].

During the course of model building, especially if evaluation has already began, most modelers tend to use inside-out modeling, which allows for specific components of coarsely summed up services to be decomposed into a series of components that reflect the desired performance characteristics more appropriately.

2.2 Workloads and Resources

Workloads and resources share the same evaluation model for representation, and are eventually tied together closely. While system architects tend to have precise understanding of the resource term, not all can define the workload term sharply.

An evaluation model is divided into two parts: the resource model[22] and the experiment model[23].

In this section, we will first introduce the building blocks of resource models. Then, the aspects of workload that represent the application of dynamics to the static resource model are covered. Finally, the models themselves are illuminated.

2.2.1 Terms of Resources

We have already referred to the resource term above, and we have roughly described its meaning. We define:

Definition 1 *With the term* ***atomic resource*** *of evaluation modeling we refer to a represented device that is capable of executing atomic operations. There is no need for a real world counterpart of the modeled device, it may be merely theoretical.*

[21] We did not study this empirically, though.

[22] The resource model may also be referred to as implementation model, because it represents the implementation of the system under study.

[23] The experiment model may be referred to as dynamic model, because it brings the aspects of time to the otherwise static evaluation model.

Definition 2 *An **atomic operation** is the execution of an instruction that is regarded as unsplittable*[24] *with respect to current investigations. An atomic operation is a service of an atomic resource. In an evaluation model it represents the consumption of a specific amount of time at a specific resource under that resource's execution model.*

Users who proceed according to the bottom-up modeling principle start at defining atomic resources. After that, the next layer of building blocks is modeled, realizing more abstract operations and their implementation based on underlying atomic resources and their atomic operations. These model building blocks are called **virtual resources**.

Modelers have to choose what abstraction level of the possible real world perspectives should be represented as atomic resources. [Rie99], e.g., shows modeling of high performance database systems where modeling the access granularity of single database pages is of essential relevance to the meaningfulness of the model. The database itself is subject to investigation; an abstraction from page access would thus be too much of a simplification and would necessarily lead to drawing false conclusions from the interpretation of corresponding evaluation results. For this case, the choice at what level to model atomic resources, must be made in favor of representing all individual database pages.

Another example is a distributed system, where a database system plays a subordinate role. Here, the database system itself can be modeled as an atomic resource without grave consequences on the quality of the meaningfulness of the overall evaluation[25].

Definition 3 *The term **virtual resource** refers to a model component that offers one or more operations, which are implemented by means of operations on other components. Virtual resouces thus do not have atomic operations.*

[24] The term 'atomic operation' is chosen in analogy to 'transaction' in database technology. Transactions are (complex) operations that satisfy the ACID criteria: atomicity, consistency, isolation and durability. Similar to database theory, we propose that an atomic operation may only be looked at as a whole. Even if discrete event simulation may split its execution into portions, we have no possibility to look at it in another way than an unsplittable unit of work.

[25] For further discussion of aggregation refer to section 3.2.3.

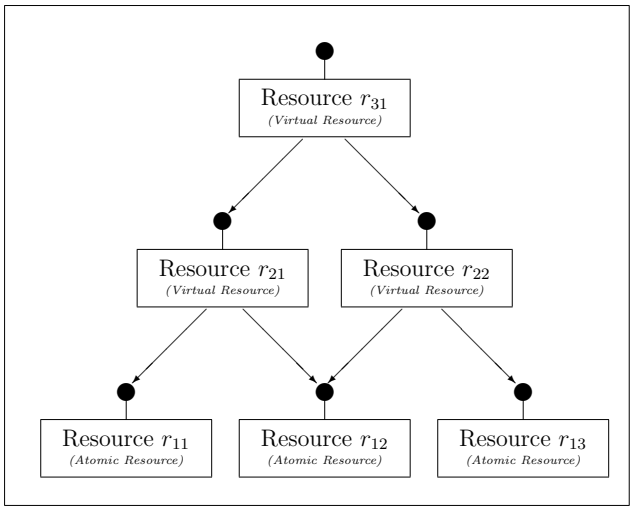

Figure 2.9: An example of a very simple resource hierarchy with atomic and virtual resources.

An example of a very simple resource hierarchy built of atomic and virtual resources is depicted in figure 2.9. Virtual resources eventually also consume time. However, they cannot do that by themselves but only by linking to atomic resources either directly or transitively.

Definition 4 *We denote the set of all atomic and virtual resources of a model as the* ***resources****.*

Generally, the expressive power at the level of atomic resources is considerably lower than at the higher levels of virtual resources. We often build resources and services that merely consume time in a resource specific way. E.g., if we encode a CPU at the level of atomic resources, we typically offer one service representing CPU consumption for a specific amount of time[26]. The possibly complex operations that cause CPU consumption have to be encoded on the level of virtual resources, where much more expressive power is available for model builders.

Definition 5 *The operations of a resource are called its* ***services****.*

[26]The amount of CPU time needed is typically passed as parameter, so the service interface is just the CPU consumption along with an 'amount' paramenter.

Definition 6 *A **resource type** is a type definition for resources as known in common programming languages. It defines the properties and behaviour of some resources in general without being specific on connections and relations, capacities, and other values varying from instance to instance. A part of a resource type definition is the service interface definition and service implementation.*

As a type is only an abstract definition, instances of that type must be built into the resource model representing the real-world objects to be modeled. The way of incorporating instances of a resource type in a resource model is to build resource pools.

Definition 7 *A resource pool is a group of resource objects of the same resource type, at the same location, exposing the type's service interface. Service requests — incoming workload — is addressed to the pool and internally assigned to one of the resource objects for execution.*

This definition allows for modeling different task assignment policies within a pool. Although many of the pools of the resource model will have a cardinality of 1, i.e. there is exactly one resource in that pool, the pool concept offers a handy manipulation point for simple capacity variation experiments.

When a workload element is assigned to and then processed by a resource, it is said to be consuming the resource.

Definition 8 ***Resource consumption** is the occupation of a resource by a workload element for a specific time. **Strict resource consumption** is the period or series of periods for which the workload element is essentially being processed[27], while **weak resource consumption** refers to the period between insertion and either completion or experiment termination.*

[27] In say the workload element is 'in execution', cf. section 2.3.3.

2.2.2 Terms of Workloads

The workload is a counterpart to the resource. It is applied to a resource and causes the resource to perform an operation and to consume time, either by itself, or by delegation.

Definition 9 *We define the term* ***workload*** *as resource consumption. The resource is in use for a specific amount of time. It is the resource's execution model that defines whether different workload elements consume a resource in parallel, sequentially or in some mixed way.*

Definition 10 *The* ***execution model*** *of a resource defines in what way workload is processed for that resource. The two basic types are parallel processing and sequential processing (sometimes called* `SHARED` *and* `EXCLUSIVE`*).*

Resources working with the parallel processing execution model schedule all workload elements for execution 'immediately'. The resources' processing speed is in many cases split amongst all elements in operation. We roughly assume that for n elements, each element is executed at $1/n$ of normal processing speed. For example, a CPU operates n processes at $1/n$ of normal CPU speed, in a very much simplified model.

Sequentially processing resources only process one workload element at a time. A new workload element is entered at the resource's queue and is only scheduled for processing if there is no previous element in the queue and if the resouce has no other workload element in process.

Of course, mixed forms of these two execution models are widespread. For example, it is common practice to define a capacity limit for parallel execution. The workload elements in execution are then executed in parallel, as denoted above, while the ones exceeding the capacity for the moment are held back in a queue.

In many cases, parallel execution leads to a general reduction in processing speed. Imagine a database system that typically operates many requests concurrently. From the simulation perspective we are inclined to assume the parallel execution model. As a matter of fact, however, a database system does many of its operations sequentially. Especially, switching from one context or transaction to another comes at the cost of additional operations,

which we call administration overhead. For n operations we then assume that processing speed is significantly less than $1/n$ of normal speed.

Of course, these processing speed phenomena must be specified as part of the resources' characteristics.

Definition 11 *The term* ***queue***[28] *refers to the buffer in front of a resource's processing that stores workload elements, which have arrived and have been accepted, until they can be processed. The order in which stored elements are passed to processing is referred to as* ***queueing discipline***[29].

We know different queueing disciplines:

FIFO first-in-first-out: The workload elements are passed to processing in the same order as they have arrived at the queue.

FILO first-in-last-out: The workload elements are passed to processing in a way that the most recently arrived element is always the first to be scheduled.

RANDOM The workload elements are passed to processing in random order.

PRIORITY The workload elements are passed to processing according to their priority. A sub-discipline is often needed to define the order of passing elements of the same priority.

Typical execution models for often used resource types are:

- A **CPU** is typically modeled as `SHARED` thus works in parallel execution model. Experience shows that a capacity limit should be defined in any case. Otherwise, overload by a huge amount of workload elements would lead to an evaluation state with almost no element finished in processing.

[28] Cf. [PW93] for a short introduction to Queueing theory.
[29] Cf. [PW93] for a short introduction to Queueing theory.

- A **disk** or a **database** is typically modeled as `EXCLUSIVE` and thus has a capacity of 1 operation. This is typically only useful if the exposed services are at the level of singular write or read operation. Higher level services should then be modeled by means of virtual resource definitions.
- A **communication network** in our assumptions has a capacity of 1 and shall also be modeled as `EXCLUSIVE`.

Apart of these rather fundamental system components, other elements can also be represented a the level of atomic resources:

- A **CPU farm**, e.g., can be modeled as pool of resources. If there are, for example, 50 CPU units in a CPU farm, the first 50 workload elements can be processed at the same speed, namely the normal speed of 1 CPU. Only the 51^{st} would be scheduled on a resource already occupied. That resource's speed per workload is then reduced to 1/2, while all 49 other resources process their workload at full speed.
- Also, a **web server** or an **application server** can be represented likewise on the level of atomic resources. A `SHARED` execution model would probably be appropriate for this case. The same remarks are also valid for **server farms**.

2.2.3 Evaluation Models

As we have mentioned, our evaluations use models that consist of both an experiment model and a resource model.

Definition 12 *The **resource model** contains all the implementational aspects from resource types and services throughout individual resources and resource pools. As resource definitions include service implementations, relative*[30] *aspects of behaviour are also covered.*

[30]With relative aspects of behaviour we mean definitions of reaction if a workload element arrives at that resource or resource structure. It is our understanding that the resource model covers structural aspects of behaviour much like a program covers structural aspects of a process. Vice versa, like a process needs a program, a workload element needs a resource structure to together uncover their dynamics.

Definition 13 *An **experiment** is an application of the experiment model onto a resource model. The experiment includes experiment means, such as simulation; experiment parameters, such as speed factors; probability distributions, or self adaption measures; and experiment boundaries, such as durations.*

Definition 14 *The **experiment model** defines the basic characteristics of an experiment. This includes the representation of external workload*[31]*, the observed values, and possible starting points for self adaptions*[32]*.*

Definition 15 *The **observed value** is a statistic value that serves as indicators for system performance. The term **key indicator** is used as a synonymon for observed value.*

The prototypical observed value is the number of external workloads processed during an experiment period. Others may be: The number of workloads for a specific resource, the average waiting time of a workload element between insertion and completion, or occupation and usage ratios for resources.

Definition 16 *An **evaluation model** is a combination of a resource model and an appropriate experiment model. In addition, it defines the strategies of self adaptions as offered in the experiment model.*

Definition 17 *An **evaluation** is a series of experiments on the evaluation model with either the resource model or the experiment model modified for each individual experiment. Its aim is to gain information about the observed values and to analyze differences as a consequence to model modifications.*

[31] The external workload is sometimes referred to as 'expected user workload' or '(expected) amount of requests to the system'.

[32] Self adaption will be explained in section 3.2.3. In any case, self adaption modifies some definitions of either experiment model or resource model; the definitions that may be modified, must be declared as part of the experiment model along with some boundaries that ensure that even the modified model represents the system under study with only reasonable deviation.

2.3 Foundations of Workload Modeling

[Rie01] introduced formal aspects of workload modeling. We will introduce an analogue representation of workloads and resources. Our definition consists of formal model construction[33] and of formal experiment modeling[34]. While the model covers the structural and implementational aspects, the experiment only makes it useful for the observation of its dynamic behaviour.

2.3.1 Model Structure

To define a model's structural properties, we introduce the set of **resources** $\mathbb{R}$ and the set of **services** $\mathbb{S}$ with:

$$\mathbb{R} = \{r_1, r_2, \ldots, r_\varrho\} \tag{2.1}$$

$$\mathbb{S} = \{s_1, s_2, \ldots, s_\sigma\} \tag{2.2}$$

Further, we define the set of atomic resources as introduced in definition 1 and denote:

$$\mathbb{R}^a \subseteq \mathbb{R}, \quad |\mathbb{R}^a| > 0 \tag{2.3}$$

and the set of virtual resources:

$$\mathbb{R}^v \subset \mathbb{R}, \quad |\mathbb{R}^v| \geq 0, \quad \mathbb{R}^v := \mathbb{R} \backslash \mathbb{R}^a \tag{2.4}$$

The sets of resources have the following relationships:

$$\mathbb{R} = \{r | r \in \mathbb{R}^v \vee r \in \mathbb{R}^a\} \tag{2.5}$$

and

$$\forall r \in \mathbb{R} : r \in \mathbb{R}^a \Longleftrightarrow r \notin \mathbb{R}^v. \tag{2.6}$$

The type and the location concept will be introduced below. Finally, a resource has some processing characteristics, such as a workload acceptance mechanism, a queue for incoming workload elements controlled by a specified

[33] Formal model construction defines the terms and ways of building a model. Model definitions consist of resources, services and some resource or service mechanisms.

[34] Experiment modeling consists of specifying the experiment terms – such as duration, observation points and observed values – applied workload, and applied self-adaption mechanism.

queueing discipline, and a processing capacity. These will not be looked at at this formal level.

Resource types $\mathbb{T}$ are groupings of equal resources from the perspective of implementation[35]. Resource types are like patterns for resources that define their structure and implementation but not instance properties such as capacity or location or run-time properties such as the specific workloads in or due for execution.

We define the *type* function:

$$type : \mathbb{R} \mapsto \mathbb{T} : r \to t \tag{2.7}$$

and the resource type set:

$$\mathbb{T} = \bigcup_{\forall i} t_i, \quad \text{with} \quad \exists r \in \mathbb{R} : type(r) = t_i \tag{2.8}$$

Inversely, we say

$$instances : \mathbb{T} \mapsto \mathbb{R} : t \to \mathbb{R}' \tag{2.9}$$

with

$$\mathbb{R}' \subseteq \mathbb{R} \wedge \forall r \in \mathbb{R}' : t = type(r) \tag{2.10}$$

in the typical case it's even $\mathbb{R}' \subset \mathbb{R}$.

The **location** of a resource refers to its geographic or logic position within the context of the observed system. We denote the set $\mathbb{L}$ of locations as

$$\mathbb{L} = (l_1, l_2, \ldots) \tag{2.11}$$

Each resource has exactly one location.

$$location(r) = l \in \mathbb{L} \tag{2.12}$$

$\mathbb{T}$ and $\mathbb{L}$ are not related to each other, as

- a type has no location, only its instances
- a type can have instances at different locations
- there can be resources or resource pools of different types at the same location.

[35] Cf. [Pie02] or [ASS96] for introductions to type systems in formal languages.

Each resource can only be either atomic or virtual. This characteristic is given at modeling time and cannot be changed at run time[36]. Besides that, resources have the following properties:

- a resource type $t \in \mathbb{T}$ (cf. equation 2.8), and
- a location $l \in \mathbb{L}$ (cf. equation 2.12); the location may also be a geographic extent, for example for the case of a communication network.

To simplify models with equal resources of the same type at the same geographic location, we introduce the notion of **resource pools** $\mathbb{P}$. We denote the set of $\mathbb{P}$ as

$$\mathbb{P} = \{p_1, p_2, \ldots, p_\pi\} \tag{2.13}$$

Like resources, resource pools are situated at one specific location:

$$location(p) = l \in \mathbb{L} \tag{2.14}$$

Each individual p is a set of resources such that the following holds true:

$$p = \bigcup\nolimits_{i=1,\ldots n} r_i \in \mathbb{R}'' \tag{2.15}$$

such that

$$\forall r_i, r_j \in \mathbb{R}'' : type(r_i) = type(r_j) \wedge location(r_i) = location(r_j) \tag{2.16}$$

The number of resources within a pool p will be denoted as its cardinality $|p|$. As a consequence, for every location and resource type — if there is exactly one resource of that type at that location — the resource forms a resource pool by itself, with exactly one member ($|p| = 1$).

It is important to notice that any two resources r_1 and r_2 cannot collaborate, if they are not in a co-location situation. A **co-location situation** is given, if either

$$location(r_1) = location(r_2) \tag{2.17}$$

or

$$r_1 @ r_2 \tag{2.18}$$

[36]However, we introduce the concept of structural modification or 'variation' in section 5.4.3, where such characteristics can be changed not during an experiment but during the course of an evaluation.

with

$$@ : (\mathbb{R}, \mathbb{R}) \mapsto Boolean : (r_1, r_2) \rightarrow true|false \tag{2.19}$$

Using the @ relation, we can represent geographic proximity and prevent a model from giving modelers the illusion of every virtual resource being able to use every other resource. Notice that, as some resources have a geographic extent, these are built to have a co-location situation at many locations[37].

Like resources, pools can be divided into a set of atomic pools $\mathbb{P}^a$ and a set of virtual pools $\mathbb{P}^v$, where the relationship is as follows:

$$\forall r \in \mathbb{R}, p \in \mathbb{P}, r \in p : p \in \mathbb{P}^a \Leftrightarrow r \in \mathbb{R}^a \tag{2.20}$$

Services are addressable operations of resources. They have to be declared and implemented as part of their resources' type definitions.

Each service $s \in \mathbb{S}$ refers to exactly one resource type $t \in \mathbb{T}$. We denote:

$$servicedefinition : \mathbb{S} \mapsto \mathbb{T} : s \rightarrow t \tag{2.21}$$

and

$$services : \mathbb{T} \mapsto \mathbb{S} : t \rightarrow \mathbb{S}', \mathbb{S}' \subseteq \mathbb{S} \tag{2.22}$$

with

$$\forall s \in \mathbb{S}' : servicedefinition(s) = t \tag{2.23}$$

For a service of a specific resource instance we denote:

$$s_r \in services(type(r)) \iff \exists t \in \mathbb{T} : servicedefinition(s_r) = t \wedge type(r) = t \tag{2.24}$$

The services of a resource are denoted as:

$$rs : r \rightarrow \mathbb{S}' \text{ such that } \forall s \in \mathbb{S}' : servicedefinition(s) = type(r) \tag{2.25}$$

Services of virtual resources have to use the services of other resources. We introduce the $uses^S$ function to represent that relationship.

$$uses^S : \mathbb{S} \mapsto \mathbb{S} : s \rightarrow \mathbb{S}' \tag{2.26}$$

[37] Consider two resources r_1 and r_2 representing individual computers. Typically $r_1 \not@ r_2$, so that r_1 and r_2 would not be able to collaborate. However, a communication network r_n can be in a co-location situation with both computers, such that $r_n @ r_1$ and $r_n @ r_2$.

As services are part of a resource type definition, the type of resource referred to is thus defined. However, the individual resources or pools that relate to each other are defined in $uses^P$ function[38]:

$$uses^P : \mathbb{P}^v \mapsto \mathbb{P} : p \to \mathbb{P}' \tag{2.27}$$

Combining these two reference functions yields:

$$uses : (\mathbb{P}^v, \mathbb{S}) \mapsto (\mathbb{P}, \mathbb{S}) : (p, s) \to (\mathbb{P}', \mathbb{S}') \tag{2.28}$$

where both, $\mathbb{P}'$ and $\mathbb{S}'$ are necessary targets but not in every case actually used during the execution of s at an $r \in p$. In other words, every service $s' \in \mathbb{S}'$ at $p' \in \mathbb{P}'$ is used 0 or more times during the execution of s at p.

Notice that a co-location situation has to be present for each usage, such that:

$$\forall p' \in \mathbb{P}' : p@p' \tag{2.29}$$

2.3.2 A Priori Observables

Workload is considered as usage of the modeled system in a specific way at a specific time. We refer to the set of all workloads as:

$$\mathbb{W} = \{w_1, w_2, \ldots, w_\mu\} \tag{2.30}$$

From an a priori perspective, a workload element w can be looked at as a request referring to a service s on a pool of resources p. The representation of that usage reference shall be the mapping function $\mathcal{M}$:

$$\mathcal{M} : \mathbb{W} \mapsto (\mathbb{S}, \mathbb{P}) : w \to (s, p) \tag{2.31}$$

Every workload element is created by an **originator**: $o \in \mathbb{O}$. We denote:

$$originator : \mathbb{W} \mapsto \mathbb{O} : w \to o \tag{2.32}$$

The set of all workload originators $\mathbb{O}$ consists of objects creating workload from outside the observed model and of all virtual resources. The objects outside the observed model are aimed to create and insert the workload under

[38] Under the condition of a co-location situation.

which the model is examined. They will henceforth be called **insertion objects** and denoted:

$$\mathbb{I} = \{i_1, i_2, \ldots\} \tag{2.33}$$

Thus the set of all originators $\mathbb{O}$ is defined as the union

$$\mathbb{O} = \mathbb{I} \cup \mathbb{R}^v \tag{2.34}$$

The notion of insertion objects has been chosen to represent the definitions of workload arrival patterns. The only activity of insertion objects is to create workload elements. We denote $\mathbb{W}^{\mathbb{I}}$ for the workload elements that have been created by insertion objects, or

$$\mathbb{W}^{\mathbb{I}} := \{w \mid originator(w) \in \mathbb{I}\} \tag{2.35}$$

As explained above[39], the considered workload on the observed system can be both:

- A series of workload elements each inserted at a specific time instant of the simulation period. The time instants and type for each workload element may come from a recorded footprint[40].
- A series of workload elements created and inserted according to a workload arrival pattern representing a typical pattern of system usage[41]

Events are occasions for any kind of state changes within the model. We define the event set:

$$\mathbb{E} = \{e_1, e_2, \ldots, e_\epsilon\} \tag{2.36}$$

Each event e has two properties:

1. a precise time instant in model time $t_e = t(e)$, and
2. an originator $o_e = originator(e)$.

[39] Cf. definition 9 in section 2.2.2.

[40] For a critical analysis of footprint, please refer to [Rie01].

[41] The Poisson distribution is widely accepted as the most representative distribution function for these patterns (cf. [HS94], [Hel98], or [Con98]).

As events are part of the dynamic aspects of system evaluation, most of them cannot be examined from an a priori perspective. However, some very important events are the time instants, at which a workload element is created and inserted into the system by a workload insertion object $i \in \mathbb{I}$ as introduced above.

The set of workload insertion events will be denoted as $\mathbb{E}^I$, the events of one workload insertion object i as $\mathbb{E}^i$. A priori, there is a natural temporal dependency amongst all events of any $\mathbb{E}^i$, such that we can define insertion events as follows:

$$ie : \mathbb{I} \mapsto (\mathbb{E}^I, \leq_{\mathbb{E}^I}) \tag{2.37}$$

where $\leq_{\mathbb{E}^I}$ denotes a partial order relation over $\mathbb{E}^I$, meaning the dependency amongst some $e \in \mathbb{E}^I$.

As every event e is created at a specific moment in model time $t_e = t(e)$, we define the temporal relationship of events as a relation $\leq_{\mathbb{E}}$ and thus state

$$\forall e_1, e_2 \in \mathbb{E} : e_1 \leq_{\mathbb{E}} e_2 \Leftrightarrow t(e_1) \leq t(e_2) \tag{2.38}$$

As we don't know the exact moment of an event e in an a priori observation, we refer to an index set $\Upsilon^i = \{0, 1, 2, \ldots, \upsilon^i\}$[42] as auxiliary order indication. We assume that every insertion object i produces workloads at events $\mathbb{E}^i = \{e^i_1, e^i_2, \ldots, e^i_{\upsilon^i}\}$ in ascending index order.

We can then define the order relation for each i as:

$$\leq_{\mathbb{E}} : \mathbb{E}^I \mapsto \leq_{\mathbb{E}^{\mathbb{I}}} : \mathbb{E}^i \rightarrow \leq_{\mathbb{E}^i} \tag{2.39}$$

with the full order relation $\leq_{\mathbb{E}^i}$ over $\mathbb{E}^i$:

$$\forall e^i_\tau, e^i_\phi \text{ with } \tau, \phi \in \Upsilon^i : e^i_\tau \leq_{\mathbb{E}^i} e^i_\phi \Leftrightarrow \tau \leq \phi \tag{2.40}$$

2.3.3 Workload Elements as State Machines

As workloads are the elements of impact for the modeled system, they are also subject to observation. In general, questions like "how much time does it take for a request to be processed from the moment the users press 'Enter' to the moment the desired results appear on their screens" can be answered by evaluating specific aspects of corresponding workload elements. In particular,

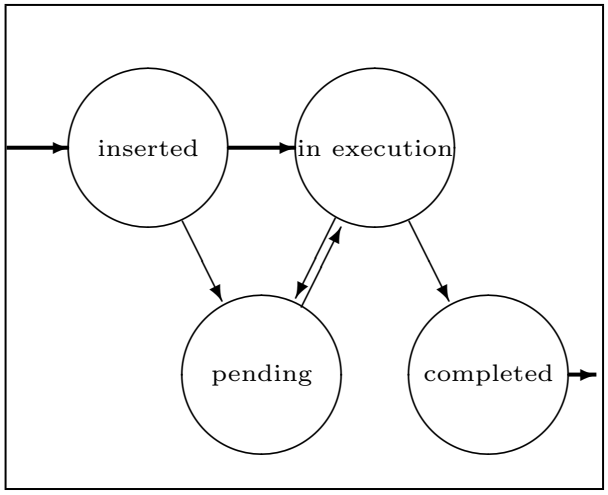

Figure 2.10: Generic Workload State Model

for the question above we are interested in the duration between workload insertion $t(e_w^i)$ and workload completion $t(e_w^c)$.

Each workload element w must therefore have specific states, as depicted in figure 2.3.3:

- *inserted* (or *created*): The workload has been created and inserted into the model at $t(e_w^i)$, but is not yet in execution.
- *in execution*: The workload element is being processed.
- *pending*: The workload element is waiting, either because of its target resource's execution policy, or because it is waiting for generated sub-workloads[43] to be finished.
- *completed*: The workload has been processed and is finished at $t(e_w^c)$.

The workload element changes its state at a specific event occurring to it. The first event is always the insertion event, whereas the last is always the completion event. The function we gives us an ordered set of events that define the state changes of that workload:

$$we : \mathbb{W} \mapsto \mathbb{E} : w \to \mathbb{E}_w \tag{2.41}$$

[42] Υ^i denotes a subset of the natural Numbers $\mathbb{N}_0$ including its order relation.

[43] Sub-workloads are workload elements created by the target resource for other resources in order to implement the called service's task.

with

$$\mathbb{E}_w = we(w) := (e_w^i, \ldots, e_w^c) \tag{2.42}$$

2.3.4 A Posteriori Observables

As introduced above[44], we denote the reference of a workload element to a service and a resource pool as follows:

$$\mathcal{M} : \mathbb{W} \mapsto (\mathbb{S}, \mathbb{P}) : w \to (s, p) \tag{2.43}$$

As pools cannot process workloads by themselves, a workload attribution mechanism is needed to select a *target resource* for every individual workload element. We denote:

$$targetresource(w) : \mathbb{W} \mapsto \mathbb{R} : w \to r \tag{2.44}$$

with

$$\forall w \exists p \in \mathbb{P}, r \in p, s \in services(r) : \mathcal{M}(w) = (s, p) \wedge targetresource(w) = r \tag{2.45}$$

Workload Descendants

For each workload element that arrives at a virtual resource, the resource's service implementation generates a set of workload elements for lower level resources. We denote:

$$generatedworkload : \mathbb{R} \mapsto \mathbb{W} : r \to \mathbb{W}' \tag{2.46}$$

The generated workload elements are processed as implicit part of processing for the causing workload element w. The processing of w can thus not be finished, unless all workload elements in $\mathbb{W}'$ are finished. Therefore, $\mathbb{W}'$ are denoted as the *children* of w with:

$$children : \mathbb{W} \mapsto \mathbb{W} : w \to \mathbb{W}' \tag{2.47}$$

with

$$\forall w' \in \mathbb{W}' : originator(w') = targetresource(w) \tag{2.48}$$

[44] Cf. formula 2.31.

Notice that for all $w' \in children(w)$:

$$\mathcal{M}(w') = (s', p') \in (\mathbb{S}', \mathbb{P}') \tag{2.49}$$

with

$$\mathcal{M}(w) = (s, p) \quad \wedge \quad uses(s, p) = (\mathbb{S}', \mathbb{P}') \tag{2.50}$$

If we look at all workload elements $\mathbb{W}''$ that have directly or indirectly been caused by a workload element w, we denote:

$$descendants : \mathbb{W} \mapsto \mathbb{W} : w \to \mathbb{W}'' \tag{2.51}$$

with

$$\forall w'' \in \mathbb{W}'' : \begin{cases} either & w'' \in children(w) \\ or & \exists w' \in children(w) : w'' \in descendants(w') \end{cases} \tag{2.52}$$

As we can see, this gives the workload a tree-like structure in the a posteriori perspective.

Definition 18 *We will therefore call the structure of an individual workload element and all its descendants the* ***workload tree****. A set of all workload trees for a specific set of workload elements will be called* ***workload wood****.*

All Events of Workload Elements

As the complete structure of workload processed by a mechanism is known a posteriori in the form of a workload wood, the performance characteristics of the observed system is tracked in the series of events of processing.

As introduced in section 2.3.2, each state change for a workload element happens at a specific event[45]. Before an experiment had ran, it was clear that the events of a workload element were in a logical order, such that the insertion event had to happen before the completion event. After the experiment, we are able to enumerate the events of a workload element precisely. We denote

[45]The question, whether the workload element itself causes an event to happen or whether an event is cause for the workload element's state change will be left open.

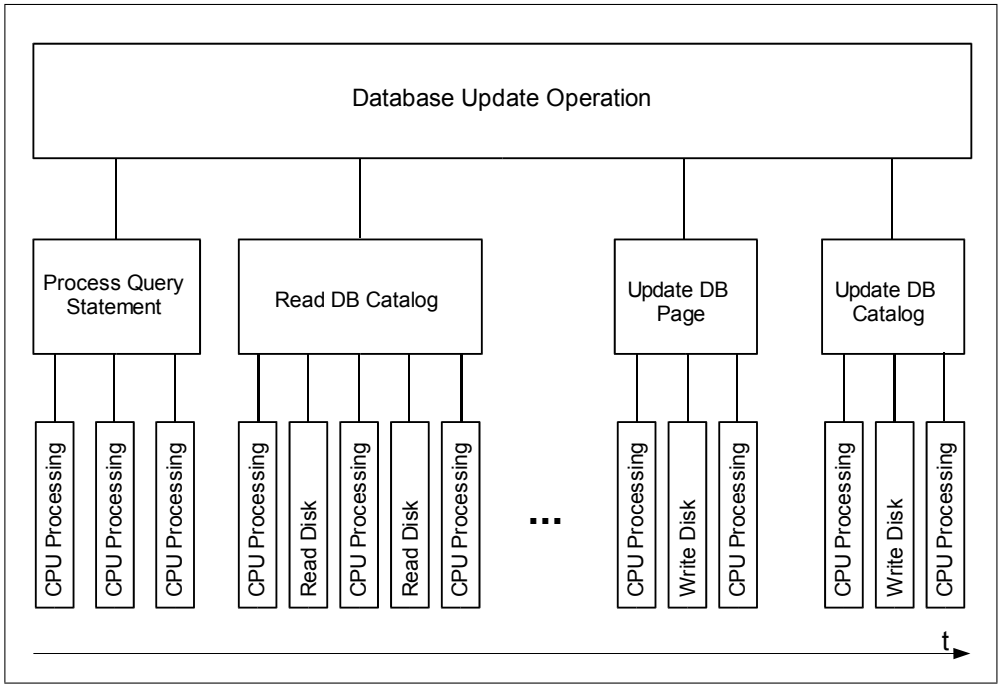

Figure 2.11: Example of an incomplete workload tree for a database update operation. The boxes on the higher levels depict abstract operations, i.e. services of virtual resources, those on the lowest level symbolize consumption of atomic resources.

$$workloadevents : \mathbb{W} \mapsto \mathbb{E} : w \rightarrow \mathbb{E}_w \tag{2.53}$$

such that

$$\mathbb{E}_w = (e^i_w, \ldots, e^c_w) \tag{2.54}$$

For every workload element w, the insertion event of all of its children is always later than the the insertion event of w; at most, the first child's insertion event coincides with the insertion event of 'w'. The completion event of all its children is always earlier than the completion event of w; at most, w completes at the moment its last child completes. This rule holds true recursively, such that:

$$\forall w' \in children(w): \quad e^i_w \leq_{\mathbb{E}} e^i_{w'} \quad \wedge \quad e^c_{w'} \leq_{\mathbb{E}} e^c_w \tag{2.55}$$

and

$$\forall w'' \in descendants(w): \quad e^i_w \leq_{\mathbb{E}} e^i_{w''} \quad \wedge \quad e^c_{w''} \leq_{\mathbb{E}} e^c_w \tag{2.56}$$

The result is an event structure similar to a workload tree, holding all events of a workload element w and all its *descendants* in an ordered and structured way, such that the first event is always e_w^i and the last is e_w^c.

Further use of the Observables

We use the model structure and the observables as a foundation for the discrete event simulation. Events do not only happen exactly at the moment of a state change of one workload element within the experiment, but also at the moment of a state change of one of the model's resources. Every event that concerns a specific resource also changes the way the resource is used. At the same time, the usage of the resource determines the way the workload elements for that resources will be processed and has thus influence on the next event that comes from the resource itself.

To draw conclusions about the performance of a system, observations of both workload elements and resources are possible, or necessary, respectively. In the following sections we refer to statistic information like, e.g.:

- The duration of procedures for workload elements.
- The number of workload elements that can be processed at a specific resource during a specific observation period.
- The time of an observation period, a specific resource had at least one workload element for processing.

The very detailed information that may be drawn from a simulation experiment could also be used for statistic analyses on a very detailed level. A possible approach would be, to discover patterns in usage or non-usage for individual resources and establish proposals for improved usage. The detailed event structures resulting from a simulation experiment would give us enough information for such statistic investigations.

However, we use the statistic information only on a very coarse granularity level. The numbers we typically use are: average throughput, average response time, average usage time of a resource, mean time of resources being processed or waiting at a specific resource or resource pool. Statistic investigations on the detailed level are not part of our study.

2.4 Evaluation Results

It is hard to tell what users of our approach are really interested in. If we look at what we have an answer for, we hope to offer what a user needs:

1. As an overview, the users are interested in the general performance behaviour of their models.
2. The next interest is probably, what influence to the maximum performance a variation at the model will have.
3. Comparisons of two or more model variations are another subject of interest.
4. An important issue is scalability, i.e. to determine how much performance can be gained by varying resource capacity.
5. Finally, there are always some individual aspects that are very important for an individual case, such as:
   - How much does the performance 'react' if we make changes at a specific configuration
   - How many workload elements have been processed by a specific resource or resource pool
   - How much internal load does a given workload element really cause?

2.4.1 Performance Maximum

The first subject of interest is probably the maximum of performance a system can get to. As the performance term is not defined sharply, we use key indicators commonly accepted as measures for performance[46].

The **throughput** is the number of workload elements that have been processed by a system during a specific observation period[47]. In our research, we use abstarct but fixed time units: A throughput value refers to the number of processed elements per time unit. Often, the throughput is looked at in the

[46] Cf. [Fer78] for an introduction to relevant performance indicators.
[47] Cf. Appendix for the term 'observation period'.

context of the workload insertion rate that refers to the number of workload elements inserted per time unit.

The throughput value can have considerable differences depending on whether a system is observed in a cold or warm state, warm reflecting the fact that the system has been run some time before the observation period has been started. The factors that have major influence on this are, e.g.:

- Loading data into a cache memory that makes access to some data or functionality faster.
- Existing workload elements that have to be processed, 'occupying' the resources that could otherwise immediately start processing the new workload elements.

The **turnaround time** refers to the duration of the period between the time a workload element was inserted and the time it is completely processed. At simulation startup (with a cold system) it is often shorter than later.

Although we do not pose general rules of interpretation for throughput or turnaround time, we generally look at mean valus for our conclusions. In a strictly sequential system, the connection between mean throughput $\mathcal{T}$ and mean turnaround time $\mathcal{R}$ for an observation period $\Delta t = t_1 - t_0$ is as follows:

$$\mathcal{T} \leq \frac{\sum \mathcal{R}}{\Delta t} \tag{2.57}$$

2.4.2 The Throughput Graph

The throughput graph depicts the rate of processed workload elements as a function of the rate of inserted workload elements. The graph's dimension is $n + 1$ for n being the number of types of external workload. If there is only one type of external workload ($n = 1$), the graph is a curve in the two-dimensional space.

As every 'function value' can only be determined by means of an experiment, the determined graph is not a representation of a real function but a plausible representation of individual measurement values. The throughput curve depicted in figure 2.12 is thus not really a throughput

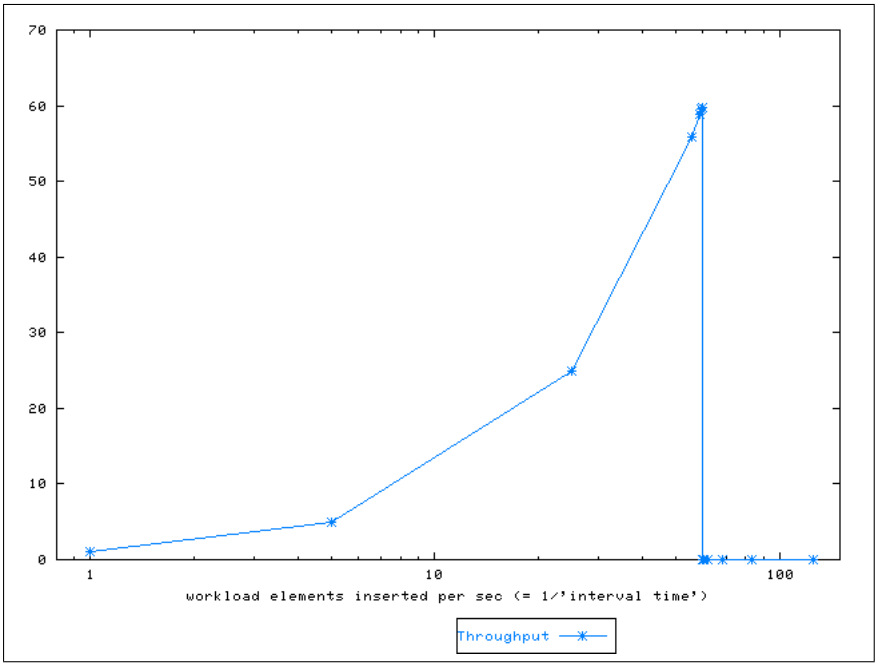

Figure 2.12: Example of a graph depicting the throughput curve as function of the workload insertion rate.

function graph, but rather it is an approximation for the limited domain range of workload insertion rates.

A characteristic of the throughput graph is that the domain values are limited either to a specific range, or even to some discreted values. This phenomenon comes from the validity rule for experiments as described in section 4.2.2. This rule essentially says that an experiment is only valid, if all conditions are met to guarantee that an experiment has relevant meaning.

If we look at the example of varying the workload insertion rate, the increasing rate at some time comes to a point, from which on experiments do not terminate validly with high probability. An invalid experiment eliminates that insertion rate from the domain set. A function value cannot be 'found', or is meaningless, respectively.

Looking at an example of two dimensional variation, especially with two continuous domain sets, the graph depicts a kind of rock landscape, for which exceeding the validity barrier at any place causes the function value to fall to 0, or 'invalid', respectively. This visual representation gives users a very good impression of the performance behaviour of its models.

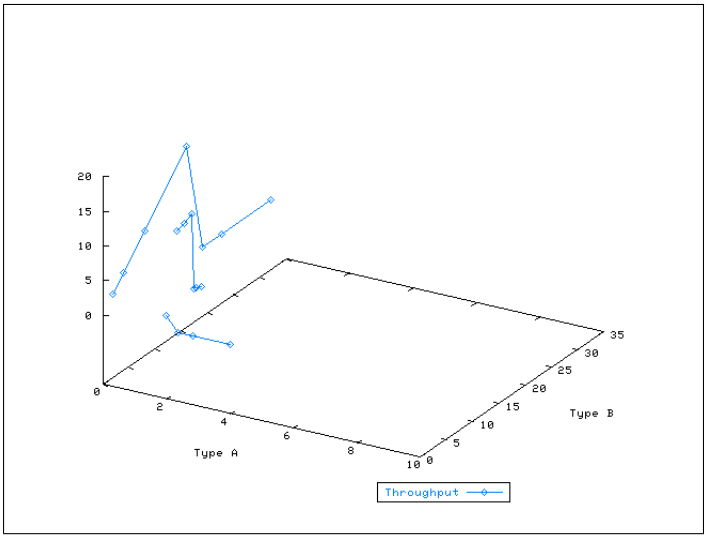

Figure 2.13: Example of a graph depicting a 3-dimensional landscape of throughput maximization as function of the workload insertion rates. This example shows very few samples to show the effects of the Warm Start more clearly.

2.4.3 Comparisons and Effects

An evaluation system looking for the performance optimum modifies the system model in predefined ways. It is important for users to know what changes have taken place to what extent, and what effects the changes were responsible for. This is what a comparison facility is useful for.

The idea is to make the effects of change directly visible. The best visibility is certainly achieved if the comparison of the performance maximum is directly depicted as a function of the change.

One-Dimensional Variation

If a model is changed only in one way[48], the throughput maximum can be represented as a function of that change. The changed value is the independent dimension, the identified performance maximum is the funcionally dependent dimension. The comparison can then be depicted in a

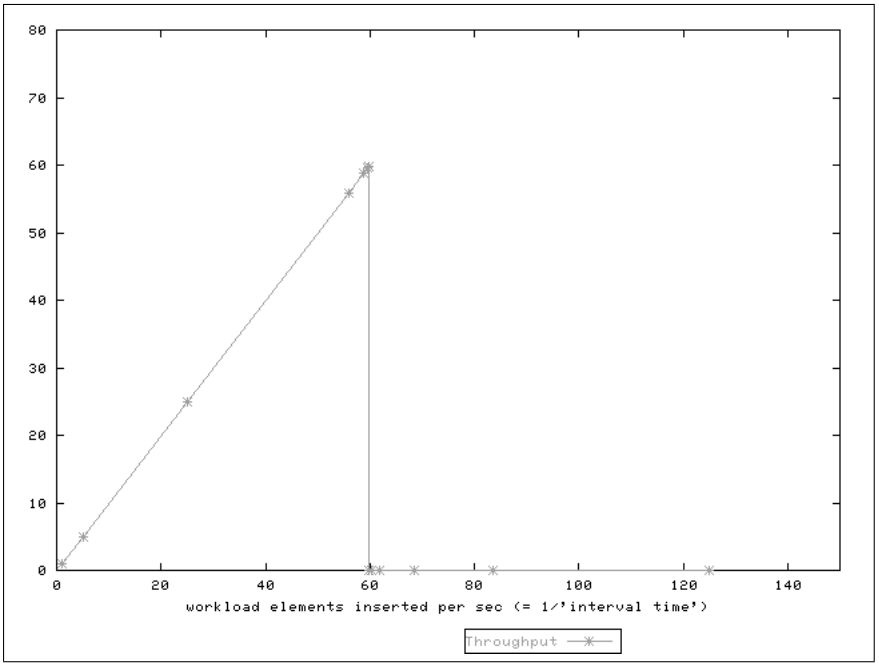

Figure 2.14: Troughput as function of the workload insertion rate.

graph approximation, if the domain values can be ordered, or in some similar representation else.

Figure 2.14 shows an example of such a functional representation. It is the maximum value the users are finally interested in. If a functional graph is not appropriate, a different form of graphic representation or a representation in tabular form is also useful.

Variations in Multiple Dimensions

If the model is changed with respect to different dimensions, due to multiple degrees of freedom, the representation for a graphic comparison is more difficult. While two-dimensional variation may still be represented in a pseudo three-dimensional diagram, the multi-dimensional comparison only takes place on the base of numeric values.

In our research, we have typically compared the performance behaviour of the modified model directly to the behaviour of the model of reference,

[48] I.e. the degree of liberty is 1.

i.e. the initial model. In this case, we compare the throughput curves of the two models.

2.4.4 Scalability Comparison

Model comparisons for scalability analysis are different from simple maximum performance analyses. While scalability analysis is presented in section 5.5 in extenso, we are here interested in what conclusion and comparison approaches can be offered.

As scalability investigations are nothing else but investigations on the performance under varying configurations — capacity values in this case — we are eventually interested in a way to compare two or more models with respect to turnaround time and throughput. However, we are not simply interested in the fact that the performance is enhanced at all, but rather in the measure of the extent to which an enhancement is possible.

2.5 Conclusion

In this chapter, we have shown the context of our research. We have introduced the notions of resources, resource pools and workload elements, insertion objects and events. Based on that, we have shown how models and experiments can or should be built.

In a part of theoretic foundations, we have introduced the formal relationships of resources, resource pools, and workload elements. We have shown how workload is created and inserted into a model, what the patterns and frequencies are, and what we know of an experiment a priori. We then showed the effect of applying workload to resources. We defined the notions of workload mapping, internal workload generation, the resulting workload tree or workload wood, and the similarly resulting event structure.

In the last part, we described what users are probably interested in, and what our approach is capable of determining and presenting. We have shown that performance evaluations must merely be based on high-level statistical data, but that users are interested in clearly presented information series of both, high-level and resource-specific measurements. We have also shown that there are different views of looking at performance, some referring to key indicators, such as throughput and turnaround time, others referring to comparative evaluations, especially in the case of scalability analysis.

Chapter 3

The Evaluation Process

Performance evaluation is best seen as a process to gain knowledge about the performance behaviour of an observed model. The underlying process model is to make sure that user interaction and evaluation activities together build meaningful iterations of experimentation and analysis. This includes the responsibility to make such advances in scientifically recognized ways.

Our approach is a hypothesis-driven process. This allows for iteratively approaching in-depth analyzed performance facts and continously improving the quality of examinations. Using this approach, users will finally end up with information that allows them make good design decisions with respect to performance issues.

In many cases, performance evaluation is part of either specific studies or of a well-defined development process[1]. Playing the role of such a part, it must be embeddable into the corresponding engineering process frame. In-depth knowledge of our approach is thus necessary to make full usage in favor of the engineering mission it is used for.

In this chapter we will introduce our proposition for a performance evaluation process that satisfies the criteria above. In addition, we give hints on how to ensure the appropriate flow of information to gain the best possible results out of the performance evaluation process. This should allow for users to recognize facts with implications on system design early enough in the engineering process.

[1]For example, the often applied 'Rational Unified Process' as defined in [JBR99b] and [Kru00].

In the first part of this chapter we compile the aims for a generic evaluation process and describe the process verbally. In the second part we take a look a the process' instruments from design throughout simulation. We show what instruments are required or useful for what action.

Finally, the third part is devoted to the requirements arising from the process model for evaluation modeling.

3.1 Description of the Evaluation Process

Processes have gained great significance in software engineering, since applying the right processes, including the application of checking and control instruments, makes sure that the right artifacts are built by the right people at the right time. But also, the relationship with the customer can be improved noticeably as engineers lead their customers to the problem solving methodologies of software engineering in a structured way and thus create comprehension and confidence.

In the following sections, we will first describe the preconditions to start from, i.e. what information in what quality must be present at the beginning of an evaluation process. Then, we will discuss the individual process steps and activities, illuminating their contribution to problem solution. Drawing the consequences, we will show what results we can gain as an outcome of the process.

Furthermore, we will make propositions for embedding the steps of an evaluation process within an embracing software engineering process. And finally, we will show the use of the process in daily life.

3.1.1 Process steps and activities

Our process description is based on a series of fundamental steps and activities. Notice that, even if each step continues on the base of the steps before, parts of the process may be ran repeatedly. We follow the idea of the spiral model of software engineering by B. Boehm[2] that essentially brings the so-called waterfall model[3] into a form of incremental interations defined and steered by means of risk assessments.

[2]Cf. [Boe88] for B. Boehm's introduction of the spiral model.
[3]Cf. [Som92] for more on the waterfall model.

Overview

We roughly identify the following steps:

1. Setup Model
2. Setup Self-Adaption
3. Run Evaluation Cycle
4. Check Results

A somewhat enhanced form is:

1. Setup Resource and Service Model
2. Setup Workload Model
3. Setup Evaluation Model
4. Run Evaluation
5. Check Results
6. Identify Changes and Re-Start Process

As a contributor to a general engineering process, the evaluation process is required to run in iterations, i.e. in activities that can be stopped and resumed arbitrarily. It is useful to look at the process as a cyclic chain of activities with the aim of gaining knowledge. A possible representation is depicted in figure 3.1.

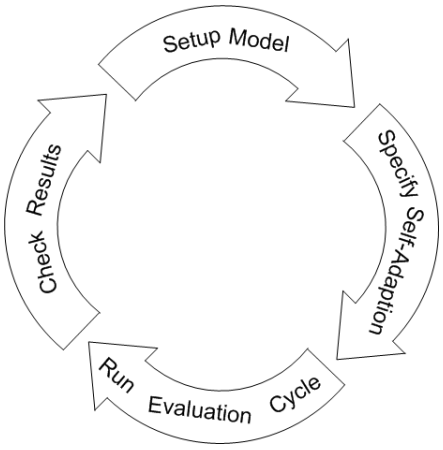

Figure 3.1: Schema of the Evaluation Cycle.

Model Construction

Model construction consists of the following steps:

1. Modeling the system architecture and design — the resource model[4].
2. Modeling the assumed workload and other characteristics of an experiment on top of the resource model — the experiment model[5].
3. Defining the evaluation and control parameters for the experiment model — the evaluation model[6]

The first aim of modeling the system architecture and design is to define the resources[7] including their characteristics. Attention should be directed especially to the resources' location[8]. To make resources addressable and usable, each resource definition must include corresponding service declarations. Finally, the logic of mutual service usage must be encoded by defining the linking between services. The latter definitions' aim is to relate each form of workload to the eventual usage of atomic resouces' services.

[4]Cf. definition 12 – the resource model.
[5]Cf. definition 14 – the experiment model.
[6]Cf. definition 16 – the evaluation model.
[7]Cf. definition 4 – the resource.
[8]Cf. section 2.1.3 – principles of distribution.

Workload modeling consists of two activities: One is modeling the workload that is inserted from external sources, the other one is modeling internally generated workload. The latter is realized in the implementation of services, while the former is to be modeled by means of insertion objects[9]. The linking between services mentioned above creates a *usage* relationship between them. As we denote that services on a higher level of logical abstraction use ones on lower levels, the linking connects higher level service implementations to service interfaces of lower level resources.

The lower services' interfaces define the service names and their formal parameters. The higher level services' implementations define the actual parameter values, the service usage – patterns, conditions, and loops – and probability distributions for probability controlled usage.

Examples of service usage:

a. An SQL query typically includes access to the meta-database (database directory) followed by many accesses to individual database pages.

   - The abstract service is a read or write operation, while lower level service is access to a specific page.
   - For most database systems, write access also includes a write operation on the database log.
   - If transaction control is included, the operation even includes request, coordination and commit included rollback operations on the transaction control monitor.

b. For the access to a web page, there is often more than one document to retrieve.

   - The whole access is divided into access of the primary document, parsing and – later – access to embedded documents.
   - A HTTP retrieval operation per se is divisible in address lookup, connection establishment, communication control, and data transfer, involving different services and nodes of the network.
   - Depending on what version of HTTP protocol is used and where the parts of the document are located, each retrieval operation requires the whole connection establishment operation.

[9] Cf. section 2.3.2

c. Some mechanisms, such as workflow tools[10], business process engines[11], or other tool control[12] facilities are even made to use and control other services on varying levels of abstraction.

d. In some cases, a higher-level service and its usage of lower-level services is explicitely used as means of abstraction. E.g. to examine the performance behaviour of a system in response to typical user behaviour, usage patterns are modeled in sessions.

Modeling the external workload is done by a specification on the level of the overall evaluation. There are different possibilities:

- The generation of workload based on a time pattern taken from an external source, e.g. in the form of a footprint[13] of an already examined system.
- The insertion of workload based on a stochastic pattern. The pattern is typically defined on the base of time intervals between insertions. The intervals can be of varying length, e.g. the typical Poisson distribution arrival pattern[14].
- Several insertion patterns in arbitrary combination of the two forms above.

A part of modeling also consists of specifying the information for the individual evaluation executions:

- To prevent simulations from running arbitrarily long, a maximum simulation time[15] has to be specified. After that time, simulation must be over or it will be stopped.

[10]Cf. [Hol95] for more on the workflow standards.
[11]Cf. [Bur01] for an introduction to Business Process Management.
[12]E.g. Tcl/Tk, Perl, or Shell Scripts as tool control facilities.
[13]Refer to [Rie01] for in-depth analysis of workload footprints.
[14]Cf. [Con98] or [Hel98] for the Poisson distribution, and [HS94] for more on Poisson distribution in load arrival patterns.
[15]Simulation Time is the real-world time a simulation experiment runs. It is sometimes also called the simulation period.

- To control the meaningfulness of a model evaluation, a target period of simulated time[16] must be specified. The choice of that target period should be done depending on the probability phenomena to be observed[17].

Model Refinement

Model refinement is nothing else than another phase of modeling. As mentioned in the overview section, an evaluation process serving as contributor to an embracing engineering process is required to both run in iterations and permit users to change and refine the resource model, the workload, and the evaluation parameters.

Model Evaluation

As explained above, the whole evaluation is thought of as a process that is ran in an iterative way. This includes user interaction and model refinement. Model evaluation itself is the part of the process that does the evaluation itself, i.e. the part that is done by the evaluation system autonomously and without user interaction.

Model evaluation runs on several layers[18]:

- The first layer is the **experiment layer**. An experiment is defined or configured according to a specific examination task. After that it is then executed by discrete event simulation tool.
- The next layer is the task layer of hypothesis tests. As described at the beginning of this chapter, we implemented our evalution system in a hypothesis-driven way. Specific evaluation tasks, called strategies[19] aim at either locating a margin value, or at proving or excluding a specific assumption. To proceed in testing its hypothesis, such a task

[16]Simulated Time is the simulated-world time, i.e. the time controlling the simulation events.

[17]As explained above, effects depending on stochastic variables can have influence on the dynamic behaviour of a system. By choosing a long enough period we can make sure that the stochastic effects appear in a statistically relevant number such that the overall evaluation takes them into consideration appropriately.

[18]Cf. figure 3.2 for the layers.

[19]Cf. chapters 4 and 5.

uses instruments[20] to create or configure, run and examine individual experiments. With a series of experiments, the task finally gains the necessary information it was ran for.

- The topmost level is the entire evaluation layer. It is the steering procedure for all evaluation tasks and follows a so-called general evaluation strategy. Its aim is to find out as much as required about the observed models.

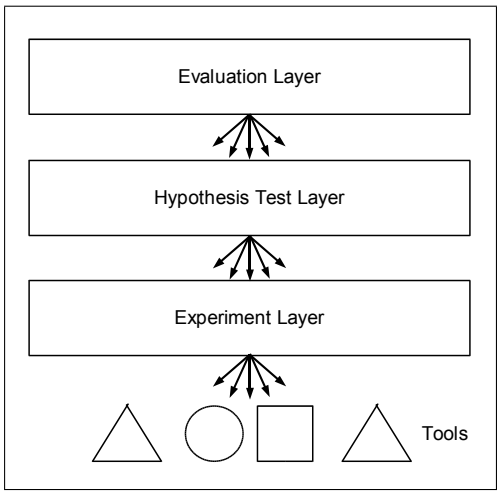

Figure 3.2: Illustration of the different layers of evaluation.

3.1.2 Embedding the Process

The evaluation process is only a part of what has to be done to build good systems. Using the information coming out of our evaluation process, system architects are in a position to make plausible decisions on architecture and design with respect to performance aspects. For example, if a system has to meet criteria of scalability — as a special issue of performance — to be flexible for future developments, designers have means with our tool to decide on the base of serious experiments, not just assumptions and guesses.

The questions to be addressed by an embeddable process are:

[20] Cf. 3.2 for more on the user instruments.

- In what form, in what way shall the performance evaluation process be ran as a part of the software engineering process?
- Which entry points and exit points must be provided?
- What kind of information has to be exchanged?
- Why should an engineer use a performance evaluation process?

The answers may appear relatively simple, although not always satisfying: The kind of information that has to be exchanged is any kind of architectural, behavioural, implementation structural, and usage-related information available. Although we aimed at establishing a modeling environment that uses building items closely related to the ones of modern engineering approaches, there is no automatic exchange of information. Such a 'tool collaboration' may be made available later, but it is not subject to our studies.

As we run the evaluation process in cycles[21], the natural entry point is the start of a cycle and the exit point the end of the cycle. The cycles can differ heavily in aim and duration. However, all cycles are assumed to run autonomously and without user interaction. As such, they are aimed at finding facts about the modeled system's behaviour that, after the cycle, are interpreted and considered as information for the system's architecture and design choices.

Aggregations are a special issue of evaluation cycles. Using aggregation techniques[22], we are able to simplify the resource model and accelerate the evaluation. Software designers use an analog technique called 'black-box' modeling[23]. However, aggregation may be used independently of black-boxing.

To use an evaluation process means to consider information valuable for the success of a solution. There are many cases for which the performance is not an issue. But distributed systems often face an unknown future; neither their usage frequency or the controlled amount of data nor required performance characteristics are always known for certain. A performance evaluation process allowing to test a model for different scenarios is thus a valuable source of information. This is especially important when scalability is an issue.

[21] Cf. 3.1.1 for more on process steps and activities.
[22] Cf. 3.2.3 for more on aggregation.
[23] Cf. [Bei95] for the concepts of black-boxing.

3.1.3 Using the Process in Daily Life

As mentioned above, the performance evaluation process is ran in cycles. The cycles are autonomous in that they run without user interaction. Because of the relatively independent cycles, there is a risk for the performance evaluation process to be only used in individual, incoherent pieces.

Our performance evaluation process is thus expected to integrate the individual questions into an all-embracing problem space that can be explored using the right techniques and strategies. Some of the techniques will be described in the following sections of this chapter. The process strategies are presented later in the chapters 4 and 5.

To be able to explore the problem space, we apply modifications, such as self-adaption[24], and learn about the effects of variation to the performance behaviour. The self-adaption techniques reflect machine-based exploration. Meanwhile, users also apply modifications resulting either from further elaboration of the architecture and design model or from other exploration interests. Doing so, they either reflect new facts, or they direct the evaluation to new possibilities of exploration. This results in an immense amount of possibilities, out of which the performance evaluation process is aimed to find the optimum configuration.

However, the evaluation process would never be accepted if it were a ponderous process with long durations. Two abilities are made to guarantee the acceptance:

- One is the possibility for users to restrict the path of exploration to lead the evaluation into promising directions.
- The other one is the possibility to run evaluations for very specific user questions. This can be achieved applying individual, specific evaluation cycles.

Application Scenarios

1. After establishing a system architecture (built of components, services, and some resource information), a project manager is interested in its performance. To evaluate it, the performance expert needs scenarios

[24] Cf. section 3.2.3.

about the system's usage. After running some initial experiments, the designer and the performance expert together identify refinements that lead to a better behaviour in situations with workload. Their dialogue lets them transform the initial architecture into one that shows better performance characteristics.

2. A customer specifies performance requirements for a new system in specific usage situations. Before giving away the mandate, he demands a study on whether or not it will be possible to build a system that meets his requirements.

3. A new distributed system with PCs and server computers is aimed at replacing a host-based solution, on which large parts of the daily operations have been ran up to date. After building some architectural approaches, the system architects discover that their architectural plot is so big that they do not have a chance estimating the new system's performance behaviour.

   Therefore, they want to design the architecture based mainly on performance criteria. They model the services and their connections, including the necessary data exchange. Using the evaluation tool, they find configurations of services and resources, which exhibit good performance characteristics.

3.2 Process Instruments

We understand process instruments as means for a part of the evaluation process contributing an evaluation ability in the form of an individual activity. Each instrument thus covers one specific ability needed for evaluating performance. As our aim is to provide a novel way of evaluating performance based on simulation techniques and tools, the process and its instruments have to find their representation in a suitable form of implementation.

3.2.1 Instruments of Modeling

As described in chapter 1, models have to be encoded and defined under different aspects. During our experiments, we discovered roughly 3 levels of modeling:

1. Modeling structure
2. Modeling external workload
3. Modeling internal usage references and workload

Of these, the first is nearest to a system model for a distributed system's architecture, while the second has to represent the expected usage. The third then is the link between the first two, making connections of the usage to its implications on the level of the elements that build the structure.

A Structure Modeling Instrument

From different modeling approaches[25] we learned that a graphical modeling tool would be best to represent the resource structure of a performance evaluation model.

The tool must provide support for:

- Modeling on the type level: Provide for an overview of what goes with what, namely what services are to be provided for what resource types and are related with what other services.
- Modeling on the resource pool level: Add information about location[26], numbers of resource instances and realizations of relationships between resource pools.
- Modeling on the resource level: This level is to model individual resources. This includes resource properties such as processing speed and others. It also includes specifications of how loads are distributed amongst the resources of a pool.

From the beginning, there must be a series of resource types that are used as atomic resource types in any model. Then, there have to be means to construct virtual resources types.

[25] E.g. UML (cf. [JBR98b], [Fow99], or [JBR98a]).

[26] Or geographic information, to be more general. Cf. equation 2.12 for our understanding of the location term.

An External Workload Modeling Instrument

To insert workload into a modeled resource system, the modelers have to specify what workload is inserted and what it refers to. Our experience showed that this is quite an easy specification, as the workload elements can be applied to modeled resource services.

In our theoretical model, there is at least one insertion object[27] that generates workload elements based on an interval pattern, such as the Poisson[28] arrival pattern or footprint data. The object does not offer a service, but has a process-like or thread-like life enabling it to be activated at self-declared moments or system events. Most tools applied for simulative evaluation allow to specify such patterns in very brief statements.

In the case of existing footprints[29], the insertion objects have the task of reading the footprint data and transforming it into loads for services. The footprint's timestamps are therefore taken and translated into model time for the determination of load generation events.

An Internal Usage Reference and Workload Modeling Instrument

Every resource offers its services. In our formal model[30], the service definition is part of the resource type declaration. We consider the offering as the resource's interface, whereas the complete service declaration is regarded as the services' implementation.

The service implementation consists of three parts:

1. The specification of its own interface including the specification of the formal paramenters[31].
2. The declaration of which services at which resources or resource pools[32]

[27] Cf. equation 2.33 – workload insertion objects.

[28] A detailed discussion of the patterns of task arrival can be found in [HS94]. Poisson distribution is described in [Con98] or [Hel98].

[29] Footprints such as described in [Rie99] can be used as appropriate patterns of usage, as mentioned in section 3.1.1

[30] Cf. definition 5 – services.

[31] Parameters are very useful for the specification of load extent. E.g. a generic resource type representing a CPU can have a formal mandatory parameter representing the requested CPU time. This allows for the CPU type to be used widespreadly, and it delegates the extent of the CPU workload up to the next higher modeling level. Probability distribution, loop control, and other programmatic constructs can also be contolled by parameters.

[32] Cf. definition 7 – ressource pools.

it refers to.

3. The specification of a usage pattern per call, including eventual parameters for the used services.

3.2.2 Evaluation Instruments

Evaluation is done at 2 different levels:

1. The elementary level of evaluation is **Simulation**. An evaluation model as described above is simulated using a simulation tool or system[33].

2. The higher level of evaluation is the application of process instruments as described in this chapter, especially the self adaption instruments described in section 3.2.3.

Simulation Instruments

When it comes to discrete event simulation, we are well-advised to run experiments using a simulation tool or system. This evaluation instrument has thus the role of controlling and executing a simulation job. However, there are other tasks necessary for the usage of simulation tools that include:

- Simulation code generation or modification to enable the higher-level processes to use simulation as an instrument.
- Job control including starting, logging and checking the result code.
- Extraction of key values from log files and result files to allow for higher level instruments to 'interprete' the results.

[33]As mentioned in the introduction, we use discrete event simulation as the base for our evaluations. Amongst other reasons, we were able to work with HIT^{TM}, a hierarchic modeling and evaluation tool that allows for integrated use of discrete event simulation technology.

Evaluation Series Instruments

While the simulation instrument's aim is to execute or run a single simulation experiment, the evaluation series instrument's task is to run series of experiments determined merely by the self adaption instruments of section 3.2.3.

This instrument is a connector between idea and technology, between strategy and execution. It allows for self adaption instruments to implement their adaptations and observe the effects. It keeps information about what evaluation has been done and what have been the results. And it closes the feedback loop to allow for self-adaption instruments to quantify the effect of the just-made modifications.

3.2.3 Instruments of Self-Adaption

A core part of our research activities was related to examining the behaviour and the change of behaviour of a system or model under varying conditions. To make this possible, there must be means to change both the model, and the evaluation, with respect to different aspects and examine the effect of these changes.

During our experiments, we identified the following basic forms of changes:

1. Model revisited by users.
2. Simplification of a model or its part by means of aggregation.
3. Variation of the workload inserted from outside.
4. Variation of the internally generated workload.
5. Structural changes to the model.

In the following sections we will examine these instruments with respect to the following criteria:

- Who — users, instruments, process — does what action with what effect?

- What is the action supposed for? What information should be gathered?
- How can actions be limited and controlled?
- What are the consequences for interpretation?

Model Revisiting

Model revisiting is the part of the process that is essentially done by the users themselves. Reasons for manual modifications and refinements of a model can be: increased demand for specific kinds of information, information about a specific construction's dynamic behaviour, and shortcomings or deficiencies in the originally examined model.

Increased demand for information may appear, e.g., after users recognize that a particular construction in the model has essential influence on the overall dynamic behaviour; e.g. if that construction part creates a bottleneck situation in workload processing. A deficiency in evaluation results can emerge when, e.g., a component does not behave as predicted or assumed in specific situations[34].

With model refinement, users are assumed to improve the meaningfulness of the model and its evaluations. They specify runtime information and behaviour in more detail.

Aggregation Instruments

Aggregation is an excellent means of model simplification. It allows for very complex resource systems to rely on a part represented by one aggregate object rather than a complex structure of resource objects and their relationships. The idea is to replace a part of a system model by one (aggregate) object that offers and uses the same services with the remainder of the model. The aggregate object must show the same dynamic behaviour as the previous object structure.

[34]Unexpected behaviour of a component is plausible on the premises that assumptions have to be made about the runtime behaviour of components not studied previously. The assumptions can often only be validated or falsified by studying their behaviour as part of an overall evaluation, or by studying them separately using benchmarking techniques.

During our research, we identified two basic forms of aggregation: Vertical aggregaction on one hand is a subsumption over multiple layers of resource structures. Horizontal aggregation on the other hand combines multiple resources of the same type — typically modeled as different physical resources.

Vertical Aggregation Suppose a resource object structure r_1 - r_2 - r_3 with r_1 being on the highest level of the three and r_3 on the lowest of the three. The vertical aggregation is aimed to examine the dynamic behaviour of r_1 - r_2 - r_3 during experiments for the entire model, i.e., with realistic workload for r_1, r_2, and r_3, and to create an aggregate object r_a with the same dynamic characteristics and the same interactions.

If r_3 is an object on the atomic resource level, the aggregate object A will be an atomic resource. If it is above that level, the aggregate object r_a will be a virtual resource. After aggregation, all used services of r_1, r_2, and r_3 will be offered by r_a and all cases of service usage from r_1, r_2, and r_3 to another object[35] are then use cases of r_a[36].

Vertical aggregation reduces the number or events and thus the simulation time by eliminating indirections. Imagine a workload w^1 for r_1 that generates one or more service usages at r_2, which in turn require an r_3 service upon each call. The reduction to r_a eliminates the workload generation and completion events of all descendants of w^1.

Horizontal Aggregation While vertical aggregation aims at simplifying multi-layered structures, horizontal aggregation allows for multiple resources of a pool to be put together in a simpler representation. The effect of this aggregation is that very fine-grain effects are generalized. Depending on the scope of evaluation, this may or may not have important influence on the system evaluation. Examples are:

1. Multiple database pages are put together in an aggregate database page pool. While the mere access to an individual database pages is more or less serialized in any database system, and as a consequence has to be represented so, the effects such as locks for specific database pages

[35] I.e. internally generated workload that does not remain in the group of r_1, r_2, and r_3.

[36] Notice that if r_a is an atomic aggregate resource, there cannot be service usage of another resource, i.e. atomic resources cannot create any load for other resources.

cannot be represented on the page level. Probabilistic mechanisms[37] might help in building a model part with a close-to reality behaviour; it depends on the scope of interest though, whether such a aggregation should be done or not.

2. Like CPUs that can run multiple processes at the same time, many atomic resources allow for multiple workloads to be processed concurrently. Horizontal aggregation simplifies the aspects of concurrent execution[38]. The workloads processing is simply regarded as parallel between two events for that resource. For a CPU object that means that it simulates the execution of n processes in parallel with a speed of about $1/n$ of full speed.

Horizontal aggregation drastically reduces the number or events and thus accelerates simulation. We say that it improves the simulation quotient. If simulating a CPU, all aspects of time slicing can be ignored; only the insertion of a new workload element or the completion of one in progress create events relevant for CPU simulation. Notice that this is only useful if we are interested in the behaviour of higher level tasks. If the CPU tasks themselves are to be observed, such a simplification would be inappropriate.

External Workload Variation

External workload variation is an instrument of self-adaption that allows to learn more about the dynamic behaviour of a system. The frequency of workload insertion is changed either by changing the pattern of workload generation or by applying a factor to a workload footprint used as workload insertion time schedule.

[37] E.g. the observation that in 5% of all database write cases a page is found locked – and the write operation has to be delayed – can be represented statistically correct in an aggregate object. If, however, that locking concerns a specific page that has effect on a specific higher-level process (or load), the aggregation may lose the ability to differentiate and represent that effect appropriately.

[38] Examples for aspects of concurrent execution are: time slicing, priority based scheduling, and phenomena of locking, as mentioned above.

Internal Workload and Parameter Variation

The instruments of internal workload modification and parameter variation facilitate the study and comparison of different service implementations but with unchanged structures and service relationships. More specifically, it is not the structure of mechanisms that is subject to changes[39], but rather the probabilities, loop parameters, and the conditional workload generation parameters are changed.

Some examples for clarification:

1. A specific service implementation contains a loop statement to execute a block of statements on the average of 5 times. This numeric specification can be varied arbitrarily. It is the user's task to restrict that variation. The mechanism of self-adaption would replace the 5 by a 6, and later by a 7 or 6.5 or by a 4 to study the effects of that change on the performance behaviour of the overall model and the changed object, respectively.

2. Another service implementation contains a statement for conditional workload generation of, say, 20% for a lower level resource's service. This means that in one out of five times the implementation is executed, a workload element for the lower level resource is generated. The self adaption mechanism changes this probability parameter and observes the effect.

3. As mentioned above, the specification of parameters allows for resource definition to expose very generic services. A CPU resource type, e.g., can expose the size of an operation[40] as formal parameter. Each object that generates CPU workload must specify that workload element with the corresponding parameter. The self-adaption mechanism changes this parameter and observes the effect of that modification.

Variation in Model Structure

The last instrument of self-adaption is the change of model structure. Specifically, such an instrument is able to test the effect of varying some

[39] Expressed in terms of programming theory that means that the algorithms themselves remain untouched for what concerns the programming statements.

[40] Or the duration of a request's processing time, respectively.

parts of a model by inserting one configuration out of a selection of possible configuration alternatives. Examples for pattern replacements are:

- The access to a file or database server is enhanced with a cache mechanism with fixed or even varying hit ratio.
- In a client/server solution, functionality is migrated from the server node towards the client node or vice versa.
- In a client/server solution with multiple servers, functionality is moved from one server to another one.
- In a client/server solution, the singular server is replaced by a group of servers which must be synchronized.

3.2.4 Instruments of User Interaction

For the performance evaluation process, we can identify the following activities that have to be covered mainly by users:

1. Modeling
2. Controlling the evaluation, and
3. Interpreting the results.

The evaluation process must offer instruments for each of these activities. During our research, we were able to recognize these instruments:

1. A **modeling tool** that allows for users to encode, modify and administer their models, experiments, and evaluations in the way defined by our approach. The resource model and experiment model definitions include the definition of the resources and services, the internal workload generation, and workload insertion[41].
2. A **tool for evaluation control** that facilitates the following operations:

[41] Cf. equation 2.33 – workload insertion objects

- Control of individual simulation runs
- Control of simulation series
- Control of all self adaption mechanisms

3. A **result preparation tool** that facilitates the interpretation, supports inquiries of specific facts and circumstances and clearly correlates the achieved results with the model and control parameters.

3.3 Process Implications for Performance Evaluation Modeling

3.3.1 Implications for Evaluation System Autonomy

As it is our aim to offer raw-power system performance evaluation as a complement to expert knowledge and intelligence, it must be possible to run an evaluation with a minimum of user interaction. To accomplish that, it is mandatory to choose evaluation strategies, evaluation instruments, and to set constraints to the chosen strategies and instruments while building the evaluation model.

- For *external workload variation*, the range of variation to be considered must be specified. Also, if different insertion objects create workload with different patterns, the users must declare whether the variation will have to happen synchronously for all, or whether it should be persued individually for each of the workload sources[42].
- For *internal probability and parameter variation*, the values to vary must be indicated. Also, ranges or constraints for variation must be defined for each indicated value.
- For *structural replacements*, the involved resource objects must be identified, and possible replacement patterns must be defined.
- For *aggregation*, the group of resources that may be aggregated must be identified, and typical workload situations must be encoded in a way that allows for the evaluation system to quantify the effects and qualify the appropriateness of the aggregation.

[42]Synchronous variation means, all loads come at equally shortened intervals, while individual variations allows for studying the effect of changes to individual load patterns for the whole system model performance.

3.3.2 Evaluation Model Evolution and Previous Experiments

As mentioned earlier, our evaluation process is ran in hypothesis-driven cycles. These cycles can be executed as individual series of experiments or as part of an overall evaluation strategy. Individual experiment series may be required by system architects to find out specific facts for the system engineering process. The overall evaluation strategy, on the other hand, is aimed at finding the essential finding using the best possible techinques with as little interaction with users as possible. All cycles, whether individually demanded or ran by an overall process have the mission to find facts and identify knowledge about the target system's performance behaviour.

The gained knowledge must be re-considered for changed circumstance. E.g., if the system model of the target system is changed, the changes have to be reflected in the evaluation model (mainly in the resource model) and knowledge gained to date must be examined. The gained knowledge may lead to other system design decisions, which in turn lead to other evaluation models and to need for other knowledge.

Thus, although the target system is steadily being changed, we want to see our evaluation process as integrated process enabling us to make consistent and detailed statements about the system's performance. It is thus responsible for making information of earlier evaluation cycles accessible for later model versions.

This is best seen for an aggregation example:

- We want to simplify a model by means of aggregation to make the execution of experiments faster. In the meantime, the performance behaviour of the aggregation object should be as close as possible to the object structure before. The aggregation object r_a is made to exhibit the same behaviour as the objects before.

- If doing the invese operation, i.e. model refinement by transforming a simple resource into a group of resources, we typically do that to dive more into detail. We want to learn more about the role of the previously composite object, not to have the group reflect exactly the composite object's performance behaviour. Not maintaining the knowledge but enhancing it is the scope of such an activity.

- If, finally, other examinations involve an aggregate object, we have to look at the effects of these examinations on the workload for the aggregate object. We have to make sure that the aggregation is still valid and useful with respect to its quantitative criteria.

It is the evaluation system's mission to keep the determined knowledge togeter and valid. Although we have not deeply examined techniques to allow the process to do so, approaches from unit testing[43] have shown to be promising. The idea we followed was to make individual test cases applicable again and again to make sure that old tests also hold true under new conditions.

[43] Cf. to [Lan04] for JUnit as an example of unit testing.

Chapter 4

Principles of Process Implementation

Machine based performance evaluation is expected to find results to some performance questions efficiently and comprehensively. It must be efficient in order to be applied as an effective tool in system engineering. Accordingly, comprehensive experiment paths have to be followed to gain confidence in the results.

As mentioned in chapter 3, we look at the evaluation as a hypothesis-driven process. It consists of cycles, each one starting with a hypothesis statement and ending with the results of hypothesis test activities. Each cycle has to determine an answer to a question, which aims at making meaningful statements about the performance of the investigated system, and aids at making engineering and design decisions.

Of course, the questions have to be asked in an ordered way, and similarly, the answers to a series of questions should be determined in a logical sequence. We therefore introduce the notion of strategies, used as a base for systematically executing evaluations.

Before that, we present some of the questions, which may appear prototypical for many kinds of performance evaluations, and for which we will look for answers later on. We show that there are connections between the questions and the corresponding answers.

In section 4.2 we will introduce some basic concepts in the context of the questions and strategies. In the third section we will present some of the

strategies, introduced systematically and described in a comparable way. In addition, we show what results can be expected from each of the strategies, and how the results are best shown. A more detailed look at the strategies will follow in chapter 5.

The last section will re-visit the user's perspective on our performance evaluation system.

4.1 Questions of Realization

In general, a performance evaluation system may be expected to answer questions of a very broad range. To deal with the questions of those who may use our performance evaluation system, we need to know what are the typical questions. In this section we look at questions recurring in situations of general performance evaluation. In order to profoundly constitute these questions and also to profoundly elaborate answers and comments, the questions are embraced in categories.

4.1.1 Question Phrasing

As we have mentioned in chapter 2, performance is not a sharply defined term. A question, like *'what is the maximum performance the modeled system is able to show?'* cannot be answered directly. Instead, we use performance indicators[1] to measure performance, and therefore ask the questions in corresponding terms.

The question above could be translated to: *'what is the maximum throughput for a given resource model, if the model is assumed fixed and the size of workload streams variable?'* Of course, this is a very verbose question and users would typically not ask it this way. However, users must be clear about two things:

- What is fixed and what can be variable?
- What are the indicators and possible threshold values to be observed?

[1]Performance indicators are often referred to as key values. Typical indicators are the throughput and the turnaround time.

4.1.2 Categories of Questions

We chose to categorize the questions and answers with respect to the item that is varied for the evaluation. This coincides with the strategy implementation. Other possibilities include user requirements and discrete or continuous variation spaces.

The following are our categories of questions:

- Questions about the raw performance of a system model
- Questions about effects on the performance if the model is changed
- Higher-order performance questions, including scalability issues

Questions regarding raw performance: Questions regarding the raw performance of a system typically have to be posed in terms of throughput or turnaround time. For these kind of questions, we assume the system architecture to be invariable, while the incoming workload can be varied. This is done in order to have varying amounts of workload elements inserted into the simulated system model.

There is a difference between the case, where there is only one kind of workload to be varied, and the case, where there are many variable kinds of workload.

One kind of workload: In the simple case, there is one kind of workload. For this case, the maximum throughput theoretically coincides with the maximum workload insertion rate[2]. The resource model remains stable, and the experiment model is modified in a way that the workload stream is varied in workload insertion frequency. The answer to the question is found by varying the workload stream size in a clever way to find out how much throughput can be achieved for the given resource model.

If there was more workload inserted into the system model than the system is able to process, a congestion of some form would appear. If this is true, the unprocessed elements consume much simulation time, the

[2]This comes from the validity rule that will be introduced in section 4.2.2; the rule eventually turns every experiment invalid, if more workload is inserted than can be processed. See below.

simulation takes much more time, and in the end the target simulated time is not reached. If the latter criterion is not met, the experiment is considered invalid[3] and the measured performance indicators are considered useless.

Multiple kinds of workload: If there are multiple kinds of workload, the case for evaluation is more complex. For that case, the question is typically more difficult, as the "throughput maximum" spreads in several dimensions. This comes from the fact that there is no general way of adding the throughput for different kinds of workload precisely.

If we would simply add up all finished workload elements, we would be able to determine a performance indicator for that individual mixture of workload kinds. If the mixture remains stable[4], we can find a local performance maximum for that mixture. In this case, the question is reduced to the simple case described above. However, if the mixture of workload elements should rather be variable, there is no general formula to calculate a throughput maximum.

Generally, an n-dimensional graph represents the local throughput maxima for n different kind of workload[5]. For the general question of maximum performance, the user has to interprete the resulting graph. If a weighting function is provided, the evaluation system is able to determine a maximum value for throughput based on the weighted sum. This absolute maximum is the point on the graph with the maximum function value.

In addition, if there are multiple kinds of workload, it is typically interesting for system designers to learn more about the mutual influence between the different kinds of workload. Workload of different kind may have an inhibitive effect on each other in a way that if combined, the system performs "worse" than for each category solely. Or, they may have a conducive effect such that in their combination they let the system perform better. We are thus interested in configurations of workload streams, where the system performs particularly well.

Looking at turnaround time, we typically aks for a maximum of workload that can be inserted and processed while keeping the average turnaround time

[3]Cf section 4.2.2 for more on the experiment validity rule.

[4]With a stable mixture we mean that for each specific workload type the number of workload elements of that type divided by the total number of workload elements remains constant.

[5]Cf. 4.3.3 for more on the variation of multiple types of workload.

for the individual workload elements below a specific threshold. However, the question may also be narrowed in that the constraint is only valid for specific kinds of workload. This is especially interesting for anomalies, where certain workload elements are processed normally, while others get stuck in a 'traffic jam' situation[6].

Questions regarding effects of changes to the model: Changes to the resource or experiment model have to be defined by the evaluation system users as part of experiment modeling. The possible model changes include changes of formal parameters, constant values, frequencies, and probability distribution definitions[7], as well as whole implementation procedures, or even model structures; we will look at them in section 5.4.

The numbers should not simply be replaced or varied, though. There must be plausible scenarios for the variation. In many cases, there is a limited amount of possibilities. For example, designers may use the evaluation tool to find out whether a specific optimization has reasonable effect on the system performance or not. The experimentation for this case is a variation of the resource model in two states, one with the conventional implementation and one with the optimized implementation.

Workload variation is not only required to analyze modified system models, but it can also be the driver for model variation. For example, if two models have to be analyzed to find out, which of them performs better with varying workload situations, and if the mixture of workload is relevant for the variation, the question must describe scenarios describing the relevant details[8].

A kind of inverse question is also addressable based on model variation. For example, given a specified maximum performance, the model has to be varied to identify the architecture that meets the performance criteria as closely as possible.

[6] Anomalies are not especially subject to our thesis, as we have defined our evaluations to find results from valid experiments, whereas anomalies often lead to invalid ones. This comes from the fact that the number of workload elements inserted into the system but not finished in processing ascends towards infinity.

[7] As mentioned in chapter 2, mean frequencies and probability distributions are often used as modeling means for discrete event simulation systems.

[8] An example is cost and earnings attributed to each kind of workload. With this, each specific mixture has a clear price / earning relationship, which allows to identify the desired mixture and the favorable model.

Scalability questions: Doubtlessly, scalability is one of the most important higher-order issues of performance. It looks at performance changes as a consequence of capacity changes.

Scalability questions typically include a scenario of how the system model's capacities can be enhanced. In parallel, there may also be an enhancement of requirements, such as an increased workload minimum, or an increased number of elements generating internal workload, e.g. to represent more clients for a Client/Server system.

Practical questions from the user's perspective: A performance evaluation system following our approach is not a benchmarking environment. It is not made to learn more about the performance of an existing product, but rather to learn more about composite systems in given scenarios where there is little experience.

The evaluation system is fed with the performance characteristics of individual products as well as structures, collaborations, and usage scenarios of the composite target system. User questions may include scenarios never experienced, such as enhanced processing capacity requirements or new collaboration structures.

The evaluation system thus serves users to iteratively learn more about their ideas, not about the real-world benchmarked performance indicators. It is made for experiencing and thus is to allow users to easily proceed with their experimentation and model modifications.

4.1.3 Question Examples

The following sample questions aim at illustrating the categories of questions and approaches introduced in section 4.1.2. At the same time, they should make clear what questions an expert would look at, whether it happens explicitely and systematically or intuitively.

Raw performance questions

Given a model of a simple component with only one kind of workload (usage), if workload elements are inserted at a specific pattern but at variable speed, what is the maximum throughput the component can process[9] *?*

Given a more complex model with several (independent) kinds of workload, what are maximum performance configurations if the mean turnaround time for workload elements of kind A must not exceed 5 seconds?

Given two or more kinds of workload, is it better to apply the workload elements in a mixed way or is it better to order them by kind? Is it better to run specific operations as batch operations overnight or to run them during normal operational daytime.

Model change questions

Given a Client/Server solution with default implementations for different services, what implementation change yields good performance gains? What combination of implementation changes yields the best performance?

Given a Client/Server solution with high potential of locality, how much performance gain is possible with the introduction of a Cache mechanism? If the Cache's hit ratio is variable, what is the best possible configuration?

Scalability questions

Given a Client/Server solution with 3 server nodes and 1000 client nodes, with what effort can the system be enhanced to serve 2000 client nodes with similar turnaround time?

If the capacities could be enhanced at the servers, what would be the maximum number of clients that could be served with similar turnaround time?

[9] In this case, the maximum throughput is equally the maximum of inserted workload, as more inserted workload would cause a traffic jam in the component and finally cause the experiment to fail.

Practical questions from the users' perspective:

What is the performance of a newly purchased product (hardware / software solution)? What are appropriate performance indicators meaningful to the users in this case?

Given a scenario and a number of hardware / software solutions, which of the solutions is the best one?

4.2 Basic Implementation Concepts

To understand the context of performance problem solving, we have to introduce a few basic concepts. Each of them describes an issue relevant for performance evaluation. We formalize the questions in a way that allows for tools and instruments to use and address them, and for users to apply them.

4.2.1 Degrees of Freedom

Variation is the fundamental principle in our approach of self-adaptive performance evaluation. With no variation at all, each evaluation would consist of exactly one experiment. The results are the key indicators for the model for a predefined workload situation. This allows a conclusion, whether the model can handle the defined workload or not, but nothing more.

To learn more about the model's performance behaviour and the effects of design decisions, model variation in specific ways is necessary. By defining possible variations of resource model, or of the incoming workload, we introduce so-called variation dimensions.

Definition 19 *A **dimension** refers to an item of variability at the observed evaluation model. As multiple items can be defined variable, there can be multiple dimensions correspondingly. Orthogonality is not a prerequisite for the dimensions.*

Definition 20 *All dimensions of an evaluation model together form the model's* ***problem space****. Each local or global solution*[10] *corresponds either to an exact combination or to an area of values for each dimension.*

Definition 21 *The* ***degree of freedom*** *denotes the number of dimensions in the model's problem space.*

Although a high degree of freedom may mean many possibilities of system improvement approaches, it also has its drawbacks: The more dimensions the more time an evaluation system will need for one evaluation cycle.

4.2.2 Experiment Validity

The interpretation of results, especially those that appeared as an outcome of complex strategies and instrument applications, requires in-depth understanding of statistic relationships. A process implementation is required to permanently assess its situation and to decide, where and how to continue with the followed strategy. An implementation needs an assessment function that reduces the valuation to an answer 'yes' or 'no'.

Validity condition

A good candidate for such an implementation is a criterion that assesses the validity of an experiment. We have first introduced this criterion to our research work for localizing the maximum throughput using the Cold Start protocol[11]. Using this protocol, the evaluation system is able to approach the maximum load insertion rate a modeled system can process by differentiating valid and invalid experiments. Roughly explained, an experiment is invalid if the amount of workload is so big that it prevented the simulation from reaching the defined target simulated time. This criterion is plausible since a big amount of workload elements slow down the simulation in such a way that the target simulated time is not reached within the simulation period.

[10] A solution is an answer to one of the asked questions.

[11] The Cold-Start Protocol will be described in section 5.2.1.

Application for the Variation of a Workload Insertion Rate

The application of the validity condition at varying a rate of workload insertion is trivial: When the rate is too high this means that the simulated system is charged with so much workload that the validity condition is not met. Notice that although such an experiment is invalid, the underlying simulation system still produces result values; however, these values must be ignored and not taken into consideration for any interpretation.

For invalid experiments, the simulation engine has to fight a rising number of workload elements inserted but not yet completed in processing. The more elements being processed, the more time the simulation engine needs to step ahead. As a result, the simulation is slowed down so much that only the first few workload elements are finished and contribute to the statistic results. Since these few elements were only processed at the beginning of the experiment, they reflect a result that is not representative for the whole experiment. This is the reason why results of invalid experiments should be neglected.

4.2.3 Constraints

Performance evaluations start from the definition of a system architecture model, including service hierarchies, internal workload generation and external workload insertion[12]. Typically, initial evaluation cycles start with a fixed resource model with no variation at the resource level admitted. In variation terms, we say the variation of the resource model is constrained strictly; the **strict constraint** allows only exactly one configuration. As the possible variations determine the dimensions of the performance evaluation problem space, strict constraints mean that the problem space is the same in scalability as it is for raw performance analyses.

Enhancing Constraints

To study a system with varying configurations, users have to enhance the constraints for the resource pools that might be modified. Each of these enhancements adds another dimension to the problem space of performance evaluations and thus increases the degree of freedom in system variation. An

[12]Cf. section 2.1.3 for more on workload insertion.

enhancement consists of rules that define how the variation can be realized and what consequences have to be considered.

The strict constraints are not simply removed; they are rather replaced by rules that represent the effects of implementing a change of an individual dimension. Constraint enhancements can be rather complicated, since they may have to reflect variable effects not yet considered for a strict representation. In other words, in many cases it is not appropriate to simply make one value variable instead of constant, since the resulting model would not be an adequate representation of the target system.

A well-known reason for that is the fact that specific execution models cause collateral workload. If, for the strict constraint case, the collateral workload effects are encoded fixedly within an individual service's implementation, such effects are typically also variable for the enhanced constraint case. This is illustrated for capacity in the example below, but it is also applicable for other variations.

A Simple Example of Constraint Enhancement

Let us consider the example of a CPU resource pool. Often we assume that there is exactly one CPU in such a pool, since it is easy to model. The internal workload distribution within a pool of only one CPU is trivial, since all workload elements are led to the same CPU. If we add another CPU, there is a need for workload distribution. Even if counting on modern operating systems to carry out this task, the representation of it in a model is somewhat more difficult:

1. First of all, we have to find out what kind of workload arrives at that resource pool in what frequencies, patterns, and sizes.
2. Then we need to detect how much these workload elements influence each other mutually. However, not only mutual influence has to be considered, but also collateral effects outside the resource pool can have grave consequences[13].
3. Using this information, we finally have to look at the influence the previous way of workload processing had on that mutual influence.

[13]This also includes consistency problems or protocol violations, e.g.

Some phenomena only appear when the pool capacity is enhanced, especially if there was only one resource in that pool previously. For example, concurrent access to shared data may require serialization; this may not be a problem, if there is only one CPU, but a problem may appear if two or more CPUs could disturb each other mutually. To enhance such a pool to more resources often makes the introduction of some protocols necessary to compensate the disturbance.

In database technology,[14] transaction systems had been introduced to implement coordination. Such transaction systems use locking mechanisms to prevent the system from entering a state from which it cannot be recovered[15]. Other approaches use optimistic protocols[16] to compensate the effect of errors.

The following must be considered at constraint enhancements:

- How much variation of the number of resources within the pool should be possible?
- How do arriving workload elements have to be assigned to one of the existing resources?
- Using which mechanisms do integrity preserving rules (locks, aborts and restarts of workload elements) have to be enforced?
- How much additional effort is imposed on the model due to the fact of capacity enhancement?
- How much does the ratio of the additional effort change depending on the number of resources in the pool?

4.2.4 Capacity

Especially for experiments that concern scalability, the concept of capacity has to be applied appropriately. The term capacity is used with different meaning in different areas of computer science. A vendor of a software

[14] Cf. [LD03] for in-depth coverage of database system implementation.

[15] The performance behaviour of database systems using locking mechanisms was described in [BDE+95].

[16] Cf. [Mul93] and [Tan01] for more on optimistic protocols.

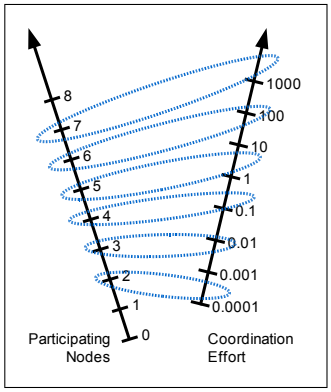

Figure 4.1: Example of the effects of a constraint. The effort for coordination (on the right scale) grows with the number of nodes participating at the solution. The dotted ellipses depict combinations admitted by the constraint, other combinations are considered invalid.

solution, e.g., may use the word capacity in an advertisement as follows: "Our electronic services have a capacity to process 300 user-requests in parallel." Likewise, a hardware vendor says: "Up to 1000 transactions per second can be processed on one server."

The close connection between performance and capacity is evident. In our work we use capacity as a given, sometimes variable information, while performance or specific aspects of it denote the information we find out by evaluations. We therefore need a concise definition of the term capacity:

Definition 22 ***Capacity*** *denotes the ability of a resource, and thus of a resource type as well as a resource pool, or of a structure of such items, to process workload elements at a specific speed. It is measured by the amount of workload elements it is capable to process and to finish per time unit.*

For simplicity, we will use the term 'resource pool' in the following sections. The combination of software and hardware, of resources and services in a specific application context is relevant for our studies. Capacity thus refers to the ability of that combination to process workload at a given speed.

Changes of a a resource pool's capacity[17] can either be done by changing the number of resources at the pool or by modifying the resource type in a

[17]Some tools use the term 'capacity' to denote the number of workload elements a resource can process in parallel.

way that resources of that type are able to process accordingly more or less workload elements.

The Issue of Capacity

So if capacity is similar to a mere component parameter, why is it important as a concept for performance evaluation? The answer is that it plays a key role in the evaluation of specific aspects of performance, especially scalability.

The ultimate question of scalability, as introduced in section 4.1.2 is: *'Given a model of fixed structure, and given the possibility to increase its capacities without limitation, where is the model's absolute performance limitation?'* At first, the answer could be that there is no limit as capacity is unlimited. However, what can be increased is always embedded in a model, its structure, its behaviour, and its components.

Capacity variation refers to individual resources or resource pools, in some cases also structures. The effect of enhancing the capacity should be that the new component consists of more processing power, i.e. it is able to process workload at a higher throughput. However, enhancing a component's capacity has often collateral effects. New resources may mean more administration workload.

A capacity change has to be analyzed carefully before modeling. All collateral effects have to be identified and represented in the system in a way that any change in capacity is reflected in the appropriate change of collateral effects. This is typically done using constraints as introduced in section 4.2.3.

Capacity Variation

A. **Increasing speed of model resources:** The most simple form of capacity enhancements is to increase the processing speed for a resource type. The raw resources' processing performance typically grows synchronously with the speed increment. There may be collateral effects due to the changed ability of processing workload[18]; however, the effects are typically less than in the case of added resources.

[18]E.g. shorter processing time for a workload element may lead to new situations, where additional synchronization effort is needed, where there was previously no need.

B. **Increasing the number of resources:** The other possibility is to increase the number of specific resources at a resource pool. At first sight, it may seem that having two CPUs in a pool instead of one doubles the speed of processing workload. However, of course, the performance behaviour of a resource pool depends on many influencing factors:

   - The first one is the granularity of tasks; one big indivisible task for a resource will not be processed any faster at a pool of more resources.
   - Another important factor is coordination and synchronization. Multiple resources often require coordination effort to keep data or control in synch, and to guarantee the integrity of the workload and the data being processed.
   - The third important factor is the workload attribution logic. Workload attribution in a fixed way imposes very little additional effort at the cost of very little flexibility[19].

   Performance experts want to do two things with a performance evaluation system: First, they want to be able to identify the resource that is most critical for the overall system performance to make a capacity modification effective. And second, they want to make all corrections, re-organizations, re-implementations, and re-structuring necessary while preserving the target system's ability to process the external workload correctly and reliably. E.g., the incoming workload elements cannot simply be attributed to any of the operating resources, but should rather behave according to its defined attribution logic.

   Scalability analysis then consists of modifications as outlined above and of model comparison with respect to performance. As a result, the

[19]Fixed workload attribution logic is explained best by an example: Assume an address management system with one database server. The system is now enhanced by another database server (on a different computer) and the address information is split in a way that all records of people with names from 'A' through 'M' go into the previous database system while those with 'N' through 'Z' go into the new database system. Address lookup workload is attributed quite simply, because every lookup operation goes to the corresponding server according to the questioned name. However, lookup information that is not based on name information will now create workload for both database systems. In addition, change operations or even transactions need to be coordinated amongst the database servers; this is an activity that was not necessary with only one server. Regardless whether it really takes more effort to resolve such a situation, new clauses of logic are required to handle this situation. Furthermore, imagine a request workload that operates highly local – i.e. it does all requests for 'A', then those for 'B' etc. (this is quite common in commercial systems); such a request workload would not profit in any way from the enhanced resource power.

modified model's performance can be compared to the original model's performance.

C. **Multiple variations, variations of different resources or resource pools:** After modifying a single resource pool and, consequently, its role for the performance of the whole system, another resource pool may turn out to be a new bottleneck[20]. Modifications are thus not limited to one resource pool, but rather multiple modifications can turn out to be useful to improve the overall system's scalability.

   Likewise, it may be useful to reapply the old modifications to the model again and to another extent. I.e., if we added a second CPU for calculations, e.g., we should also try to do it with a third one.

4.3 Strategies

4.3.1 Strategies Description Systematics

Definition 23 *A **strategy** is a generic plan to either verify or falsify a hypothesis[21], or to localize threshold values of a key indicator[22]. A strategy follows its plan by applying instruments to a model and by running an experiment on that model after each instrument application.*

Any evaluation process strategy includes instrument selection and configuration according to plausible expert behaviour.

To illustrate the aims of specific strategies, we divide them into the following three types:

- **General strategies** address the raised questions in a general way. They are general-purpose strategies in the sense that they seek solutions in a simple and comprehensive way.

[20] A bottleneck in the context of scalability means, a critical resource pool that impedes the system to scale better.

[21] A hypothesis is often defined as a key criterion, a decision criterion based on key values determined in experiments. Cf. appendix 'Terms of Performance Evaluation'.

[22] A key indicator or key value is a statistic value determined in an experiment or in a series of experiments. Typical key indicators are throughput or turnaround time. Cf. appendix 'Terms of Performance Evaluation'. A threshold value is in most cases a local maximum or minimum of a variable value given a discriminant assessment function.

Question	The subject of interest to be addressed by the evaluation system.
Explanation	An explanation of what the problem is within the context and the building blocks of this evaluation system approach.
Instrument	An indication, possibly a short discussion, of the instruments needed to answer these questions.
Control Requirements	Requirements a strategy must cover to be able to solve the adressed problems.
General Strategy	Discussion of the general approach.
Optimization Strategy	Discussion of one possible strategy that seeks for a solution in a very efficient way.
Result	What in general is the result information identified after following the strategy mentioned above .
Presentation strategy	What is a good way of presenting the results such that the users can easily interprete the results and draw their conclusions.
Application example	An example of a situation, where the questions are relevant and the strategy produces valuable results.

Figure 4.2: Elements of a strategy description and their meaning.

- **Optimization strategies** address specific problems more appropriately. They follow a more complex procedure, which enables them to seek solutions in a more efficient way, e.g. using the least number of experiments possible.

- **Representation strategies for the users** answer the quenstion, how identified results are best offered to the users such that they draw the necessary solutions.

In the following sections we will look at some of the questions that users may investigate using a performance evaluation tool. We describe the questions, the general strategy, and possibly the optimized strategy in a few words and present a user representation for each. The description will be constructed of the description elements illustrated in figure 4.2.

4.3.2 Varying one Kind of Workload

Of the questions in section 4.1, we will now look at a few particular ones that, from our perspective, represent a major part of the problem space. We will clarify the question and give an answer, in some cases by introducing an approach that will be explained later in this thesis.

Question: *'How does a system behave if we vary the incoming workload? What impact does the varying workload have on the performance? Can we use this to determine a maximum performance of the system?'*

Explanation: A part of setting up an evaluation model consists of modeling the assumed incoming workload. This is either done by defining patterns and corresponding parameters, for which workload elements are generated, or by referring to a deterministic workload stream.

- **Stochastic workload insertion** is typically based on a Poisson process[23], where workload elements are inserted with Poisson distributed interarrival times.
- **Deterministic workload insertion**, on the other hand, is a reconstruction of a previously measured or realistically synthesized workload arrival process.

The so-called insertion objects[24] implement the workload insertion mechanisms and insert the workload elements into the resource model for processing. To vary the incoming workload means to change the workload stream in a way that more or less workload is fed into the observed model. The variation is typically not done during the execution of an individual experiment but from experiment to experiment. The variation of incoming workload is realized as implementation variation of the insertion objects.

Inserting the workload elements of a bigger stream into the observed system model will have an impact on the system's run-time behaviour. The question is whether more workload, i.e. a bigger workload stream, results in a higher throughput indicator.

As in our evaluation approach we use the workload insertion objects as black-box components, variation means to specifically change a parameter determining the size of the workload stream.

[23] Cf. section 3.1.1, [Con98] or [Hel98].

[24] Cf. equation 2.33 and section 3.2.1 for more on the insertion objects.

Instrument: The applicable instruments are:

- A code modification instrument that modifies the parameters of the workload generation mechanisms, or
- A translator able to reproduce a usage footprint at different rates with respect to simulated time.

Control Requirements: To answer the raised questions, we need a strategy to determine at what speed the experiments should run. It is required for the strategy to have access to some information out of the results of previously executed experiments that allows the implementation to identify the ranges that might be interesting for further investigation.

General Strategy: A general strategy is to start evaluating at an arbitrary insertion rate (or acceleration factor). After a first experiment, we know whether the corresponding workload produces valid results or not, i.e. whether the validity condition is met or not. If so, we continue with another insertion rate higher than the previous one; if not, we lower the rate and try again. Due to the stochastic character of experiments, we cannot localize the validity limit precisely, but we can narrow the region, where it probably is. After all, we assume the performance maximum to be at the highest insertion rate that produced valid experiments.

Optimization Strategy: We have developed the Cold Start Protocol to identify the limit of load insertion a system is capable of processing in an efficient way. The Protocol is a strategy that follows efficient principles for limit localization and narrowing. It is aimed at localizing the validity limit in as few steps as possible and refining the localization afterwards. The Cold Start Protocol is defined and described in section 5.2.1.

Result: As a result, we have a series of key performance values each determined by an experiment for a specific configuration.

Presentation Strategy: The identified results can be represented graphically with the performance indicator as a function of the workload insertion rate, or alternatively, of the speed factor.

Application Example: The addressed questions are applicable as part of almost every other evaluation strategy. The presented strategy is useful not only to get a general first impression of the performance of any system or component, but also as a base component for more complex, higher-level mechanisms. E.g. an individual study of a cache mechanism allows conclusions of how such a mechanism is positioned and applied in a complex system architecture.

User Input: Variation of incoming workload requires little input by the users of our evaluation system. By defining the stream of incoming workload, the users already have defined the element that is subject of variation. For speed factors or insertion rates, they can then define ranges or sets of values, out of which one is chosen for every experiment.

Typically, such a definition is not done for two reasons: To define discrete values is contradictory to working with stochastic processes and limits the strategy possibilities; the Cold Start Protocol, i.e., is no longer applicable. And, to define lower and upper bounds for possible values in a continuous range limits the significance of an evaluation series, as maximum values are not always within the defined ranges, and values cannot always be chosen in a way that is best for the chosen strategy.

4.3.3 Varying Multiple Kinds of Workload

Question: *If a system has multiple kinds of incoming workload, which do not correlate, what effect on the system's performance does a change in speed or frequency for some of the workload types have? Is there still something like a performance maximum? How and under what premises can it be identified?*

Explanation: There are systems that have more than one kind of incoming workload, i.e. workload of different characteristics with little or no correlation at all. Different kinds of workload have different insertion characteristics, frequencies, and impact. As a consequence, they have to be modeled separately, using disjoint insertion objects, while correlating kinds of workload can in many cases be modeled using only one stream and probability based case distinction.

As the nature of the workload elements is different, adding the processed workload elements of any kind to calculate a throughput value is inappropriate. There are thus two possible approaches, which can even be combined:

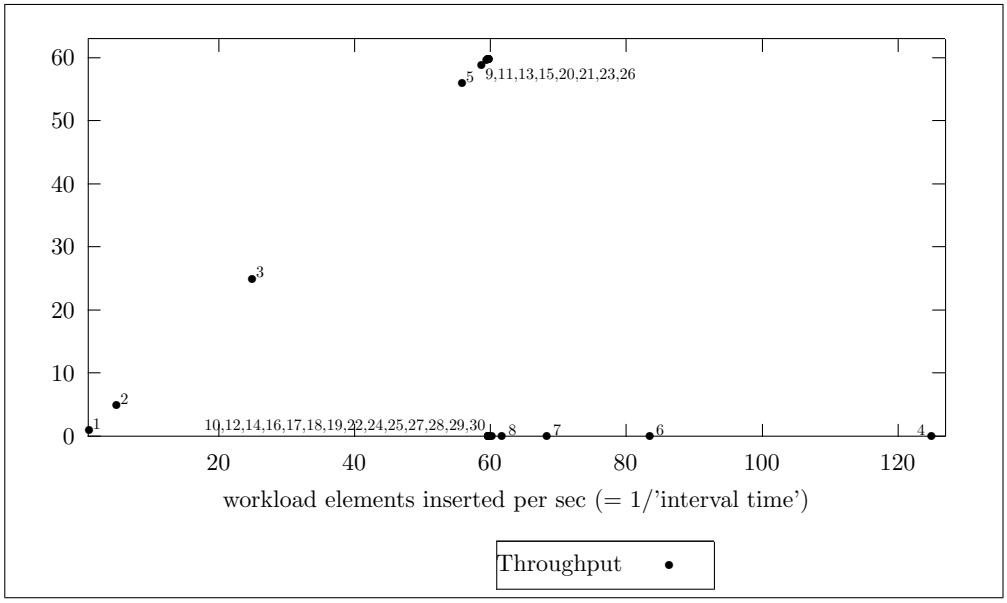

Figure 4.3: Example of one-dimensional localization of the performance maximum. The small numbers for the points refer to the experiment number in the localization series. Notice that 26 out of 30 experiments in this example are for the approximation phase of the Cold Start.

- Either the users define a formula that allows for the evaluation system to interpret the identified measurement values combined in a single functional value.

- Or, the identified values are represented in a $n + 1$–dimensional landscape. This allows for an evaluation system only to identify a maximum for the n^{th} dimension with given values for the $n - 1$ other dimensions.

Varying multiple kinds of workload thus means varying the workload insertion at insertion objects i_0 to i_n using much the same techniques as for the simple case of only one kind.

Instrument: The same instruments of workload variation applied for one-dimensional variation can also be applied here.

Control Requirements: The control requirements for multi-dimensional variation are the same as for one-dimensional variation. In addition, users may choose to influence the order of variation selection, since some kinds of workload may be more useful to find out precise results than others. E.g. an often recurring, short-living workload element is typically a statistically more calculable quantity than a long-living, scarcely appearing workload element.

General Strategy: Any strategy for multi-dimensional variation follows the same principles as those for one-dimensional variation. For n kinds of workload, all kinds mutually independent, there are n workload insertion object and thus n degrees of freedom. A general strategy is required to vary all workload streams and observe the effect of each modification. We expect it to create a general impression about the performance behaviour of the observed system. As a n–dimensional representation is applicable in all cases (n denoting the number of disjoint workload types), a general strategy is always capable of creating a landscape for reasonable ranges of all variables, i.e. with reasonable amounts of inserted workload.

The value selection for workload insertion may be set up based on the following principles:

1. To make observations of changes for individual workload types possible, we recommend the strategy to follow paths, where $n - 1$ dimensions remain constant, while the n^{th} dimension is varied at i_n.

2. Based on this path, the strategy is to identify a local performance maximum, which is essentially a one-dimensional maximum for a given context.

3. These principles are applied multiple times, while varying in another dimension, i.e. changing a second workload stream (at i_{n-1}) and iterating over principles 1 and 2.

4. The variation of the second parameter is also led to a maximum. The maximum value is identified as follows: If an experiment is invalid and the workload inserion of i_n is zero, the insertion at i_{n-1} is beyond maximum. If it is valid and the workload inserion at i_n is more than zero, the insertion at i_{n-1} is below or at the maximum. To identify the maximum, the workload of i_{n-1} has to be varied further, using the same principles as in principle 2.

5. The other dimensions have to be explored in analogy to principles 3 and 4.

The case where more workload from an i_n allows the system more throughput for workload from an i_{n-1} is called an **anomaly**. It is possible in real-world situations, especially in heavy workload situation, not only for bad but also for good implementations. However, to simulate such situations requires modeling on a very detailed level. We therefore do not investigate these phenomena in this thesis.

Optimization Strategy: Optimization strategies aim to find out qualitatively similar information with much less effort, i.e. using less experiments.

The first part consists of applying a modified Cold Start Protocol for one variation to localize a maximum value in one dimension, with the other dimensions on the value zero.

The second part then consists of angle bisection[25].

Result: The result is a set of measurement value tuples, each representing the throughput for a specific configuration of incoming workload. In their raw format, the tuples can be depicted as a function in the n-dimensional space. However, using a weighting function, a maximum for the overall throughput can be plausible.

Presentation Strategy: A natural way of representing maxima of an multi-dimensional strategy is to depict the result tuples as points in a landscape of measurements with $n + 1$ dimensions. This presentation is applicable in any case, since the meaning of each maximum point is relative. Only if the users specify a function to add values of different workload types, a maximum value can be identified and the function plotted depicting the absolute maximum as function of some of the workload insertion rates.

Application Example: The application of strategies for multiple kinds of workload is typical in performance evaluation. Overall system evaluations as well as some evaluations on the component level typically rely on its techniques.

[25] Cf. 5.3.3 for more on the Warm Start Protocol.

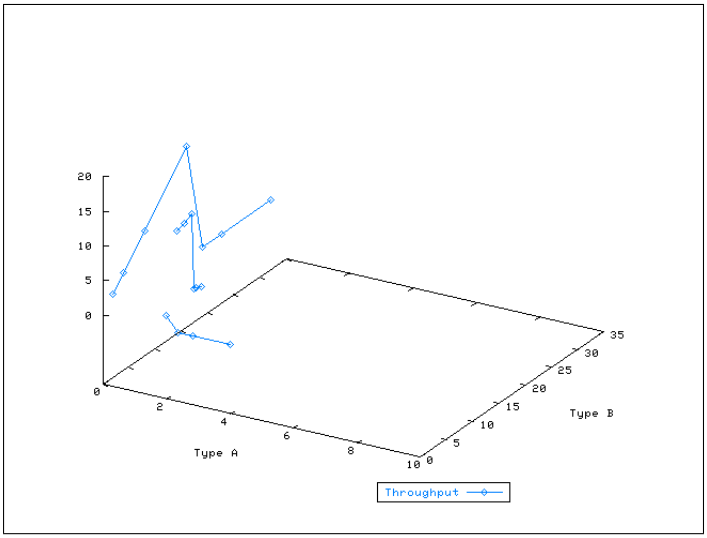

Figure 4.4: Example of multi-dimensional localization of the performance maximum. The figure shows only three evaluation paths to keep it usable, a path meaning a ratio of different workload elements. The highest point of each path is the path's performance maximum. The performance maximum sought for is the set of these highest points.

User Input: User input starts at modeling workload insertion using appropriate insertion objects. As with single kind workload variation, they can then define continuous ranges of values, or sets of discrete values, out of which one is chosen for every experiment. If specific conditions have to be met regarding ratios or thresholds of different kinds of workload, users are required to encode them as constraints[26].

4.3.4 Orders of Strategies

As introduced in section 4.1, the questions and the corresponding strategies are categorized with respect to the solution approach, i.e. with respect to the varied item. Similarly, the strategies are built in a way that some need the others to realize their implementation.

As we have seen, the maximum throughput localization is a base for most other strategies. The multi-dimensional localization is built upon a variation

[26]Cf. section 4.2.3 for more on constraints.

of the one-dimensional localization, using it from a better estimated starting point. All strategies of either parameters, implementation, or structure variation depend upon the localization strategies as introduced above.

Higher-level evaluation strategies, such as scalability analysis strategies are built upon both variation strategies and localization strategies. E.g., they rely on the variation of capacities in the scalability analysis case. Capacity variation and the localization mechanisms underneath are used as building blocks to implement how capacity changes are used and how these changes are done in good ways.

4.4 Strategies from the Users Perspective

In this section we reconsider the strategies from the user's perspective. Since the users of our evaluation system have different education and experience, they necessarily have a different understanding of performance issues. As a consequence, they also have different requirements for such a tool and different expectations of what can be done with it.

4.4.1 User Requirements

It is desirable that our users have a more or less elaborate idea of a model, and also have an information need. The aim of the user interface is to cover the information need in the most flexible way possible. Experienced users know with what strategy they can address what questions. Users with less experience need assistance in selecting appropriate strategies.

Even if users do not know the strategies well, they probably know what information they want and what variation they may require to identify the information. It is the evaluation system's task to choose an appropriate strategy.

Users of our evaluation system expect it to support them at configuring variation by proposing strategies, and by recognizing what information can be gained by applying the chosen strategies. Since many strategies use valuation functions, e.g. to determine good configurations from a range of possible parameters, the users' indication of what information is required is already a key to successful strategy selection.

The ideal for the users would be to get support with respect to possible variations at modeling time, i.e. when they construct the resource models. Unfortunately, as the used Hitgraphic tool has no notion or conception of our strategies and variations, this cannot be covered in the GUI tool.

Some variations have major consequences. For example, if the number of resources within a resource pool should be varied, there may be large-scale collateral effects with respect to the modeled system's behaviour, depending on the individual configurations[27]. The users expect to be supported by the evaluation system if choosing this kind of variation. Especially, approaches of defining constraints correctly and consistently should be supported by the evaluation system's user interface.

4.4.2 Strategy Transparency

As used in [Gro02], we understand transparency to be a characteristic of a system to offer abstractions for complex situations. The aim of the abstractions is to make usage more simple and to give occasional users access to most of the provided functionality. For evaluation system strategies, transparency means to provide a range of applicable strategies in a simple way. The more transparency, the less the user needs to know or to specify to apply strategies.

Definition 24 ***Strategy transparency*** *is the ability of a performance evaluation system to run strategy-based evaluations without forcing the user to select and configure the strategies.*

Strategy transparency is a key to understanding how the evaluation system works for differend types of users. Users with little experience in evaluating system performance are expected to use the strategies as offered. Their usage requires a high level of strategy transparency. The strategies are supposed to be selected automatically and run according to their default configuration.

Users with expert knowledge, on the other hand, are not only interested in what the evaluation tool can find out by itself, but also want to influence the evaluation to find out even more, or to find the key points even faster.

The possibilities of what experts may want to do are:

[27] This is discussed in sections 4.2.3, 5.4.2, and 5.4.3.

- To check what strategies are followed
- To select the strategies to follow
- To control how the strategies are followed (e.g. by specifying corresponding parameters), and
- To provide new strategies

Chapter 5

Process Implementation Strategies

Several strategies have been presented superficially in chapter 4. Using these strategies, a performance evaluation system is able to determine major aspects of a model's performance behaviour and to represent the results in comprehensible ways. However, to interprete the identified results concisely, in-depth knowledge of the applied mechanisms is required.

In the first section, we introduce a series of applicable strategies, ordered by their evaluation activities. It is similar to the systematics presented in section 4.3.4.

The next few sections contain in-depth coverage of the strategies as introduced in chapter 4. The second section looks at one-dimensional localization and presents the Cold Start Protocol. In the third section we look at multi-dimensional localization and the way it uses improved one-dimensional localization for itself.

In the fourth section we discuss the three identified ways of varying models. In the fifth section we eventually take a look at some strategies for scalability examinations and other issues of capacity variation.

5.1 Applicable Strategies

As mentioned in chapter 4, strategies are procedures applied to examine and validate hypotheses by issuing suitable experiments. One of the fundamental strategies looked at in this section is the localization of margin values. Localization strategies aim at locating the range of possible values of a variable, which has influence on the target system's performance, and narrowing the range considered as the validity limit. Another strategy is required to examine the effect of specific changes to the model as defined in corresponding constraints; the changes may include both, changes in implementation, and capacity changes for scalability questions.

Localization Strategies: As mentioned, localization strategies are fundamental means of performance evaluation. Using localization strategies, a performance evaluation system is able to search performance relevant 'input' values that lead to local maxima. E.g., by localizing a maximum value for workload insertion, we are able to approach a performance maximum for a pre-defined model.

In the simple case, there is one dimension for the localization, typically one kind of inserted workload. The workload insertion is varied within a range of possible values to a point, where the throughput maximum for that configuration is localized with sufficient preciseness. This strategy is explained in detail in section 5.2.

If a maximum has to be located for a combination of multiple kinds of workload, a multi-dimensional localization strategy is needed. Such a strategy is often based on one-dimensional localization. It has to produce a performance overview for local maxima at different load insertion configurations and relationships. A multi-dimensional is explained in section 5.3.

Model variation strategies: In many cases it is not enough for a performance evaluation system to localize maximum performance values. Rather, the question whether performance improvements can be achieved by modifying the underlying model, is also to be included in systematic examinations.

An evaluation system is to apply changes in ways and to extents that are pre-defined. After that, the effects of these changes will have to be

investigated. We propose appropriate strategies to use the localization strategies to investigate a modified model and to be able to establish comparison values. All proposed strategies for model variation are based in their implementations on localization strategies. Model variation is discussed in detail in section 5.4.

Scalability strategies: Strageties to analyze a model's scalability characteristics can be looked at as a special form of model variation strategies. They investigate whether a capacity enhancement can lead to performance enhancements.

Although capacity enhancements seem to be nothing more than model variations, there are differences with respect to the investigated facts. On one hand, a capacity enhancement is in many cases a rather complex operation, as many parts and parameters of the resource model may have to be modified. On the other hand, the criteria for scalability analysis are different: While general performance evaluation analyzes possible performance gains due to model modifications, the capacity evaluation looks at how much the improvement was and how much further improvement may be.

Scalability analysis seeks the answers to the following question: *"Given a resource model, workload characteristics and defined ways of enhancing the capacity, what performance can ultimately be reached if there were no limitation in capacity enhancements?"*

5.2 One-dimensional Localization Strategies

Even if we assume that the resource model is fixed with respect to the resources, the capacities, parameters, and the service implementation, there is always one degree of freedom in performance evaluation that allows us to learn something about the model's performance behaviour. The varied dimension is workload insertion, i.e. the size of the inserted workload stream. Using variation on this dimension, we are able to find out some of the model's performance characteristics.

By varying the stream of incoming workload, we are able to approach an amount of workload that is close to the maximum amount of workload the observed system can process. To do that, we need a one-dimensional localization strategy as introduced in section 4.3.2. An efficient strategy is the

Cold Start Protocol that will be introduced in the next section. Afterwards, we will also explain better strategies that can be imagined based on certain assumptions.

5.2.1 The Cold Start Protocol

Experts typically analyze the raw performance of a system or model by locating the maximum possible throughput. This strategy varies the external workload and analyzes the effect of the variation on the average throughput.

The **Cold Start Protocol** defines a strategy to determine the maximum throughput of a system model, assuming that the insertion rate for one class of workload can be varied. The strategy works in a similar way as the so-called "Slow Start Protocol" for TCP/IP that is to determine a good TCP window size[1].

The protocol assumes on one hand that a model with a rather low workload insertion rate is able to process the incoming workload appropriately. In this case, the experiment is terminated validly. On the other hand, if the insertion rate is high enough, the experiment will no longer terminate validly. Thus, if we begin with an insertion rate that leads to a valid experiment and increase the insertion rate, we will eventually cross the validity limit. Or, if we begin with an insertion rate that leads to an invalid experiment and lower it consecutively, we eventually will also cross the validity limit, this time from the invalid to the valid side.

The key to localizing the validity limit is to gain information about the range of insertion rates, where the limit is. We depart from an arbitrary starting point, e.g., one, meaning that one workload element is inserted every time unit. If the initial experiment has produced a valid result for the insertion rate one, the validity limit must be between one and ∞. To explore the yet unknown range of possible insertion rate values efficiently, we apply some procedures which are closely related with general bisection algorithms. Using them as approximation procedures, we eventually localize the validity limit by applying convergence techniques. The procedures are explained below in this chapter.

[1] TCP Slow Starts controls the congestion windows managing the number of IP packages sent into a communication session without awaiting individual acknowlegdements. It is needed to improve the overall session quality and minimization of re-send requirements on IP networks of varying stability. Cf. [Ste97]

Due to the stochastic nature of simulation experiments, the validity limit cannot be found deterministically; we make the assumption that it is a range, which we want to locate and narrow as much as possible. The random phenomena also prevent us from locating the validity limit using a formula. We therefore run simulation experiments over a sufficiently long period to make sure that the random phenomena appear in statistically relevant frequency, which allow us to examine them.

As a stable system is able to process all incoming workload elements in a valid experiment, the insertion rate and the throughput value coincide for all valid experiments[2]. The throughput maximum is then found for the highest insertion rate that leads to a valid experiment.

The Cold Start Procedure

The procedure consists of the following two parts:

1. It quickly searches a range, where the validity limit is crossed.
2. It limits the boudaries of the validity limit by means of a bisection procedure.

Quick Search for the validity limit: In a situation where there is no information about what workload insertion rates lead to a valid experiment, the search starts at an arbitrary value ι. We define:

$$\iota = \beta^{\eta} \tag{5.1}$$

with an initial value $\eta = 1$ and an almost arbitrary $\beta > 1$, typically $\beta = 2$. The searching procedure consists of incrementing η positively or negatively. If the first experiment was evaluated valid, the increment of η is $\eta_{new} := \eta_{old}+1$, else $\eta_{new} := \eta_{old} - 1$.

This procedure is terminated, as soon as two insertion rates ι_x and ι_y are found, between which the validity limit is very probable to be found. If an η_x has led to a valid experiment and η_y to an invalid one, then $\eta_x = \eta_y - 1$ holds true after this step. Figure 5.1 depicts the implementation of this first half of the procedure.

[2]Except for some minor statistic differences, depending on the counting criteria.

```
beta := 2
eta := 1
iota := beta ^ eta
target_time := 10000
exp := new experiment('xy', iota)
exp.execute()
if (exp.reached_time > 0.99 × target_time):
    increment := 1
    started_valid := 1
else:
    increment := -1
    started_valid := 0
fi
stop := 0
repeat:
    eta := eta + increment
    iota := beta ^ eta
    exp := new experiment('xy', iota)
    exp.execute()
    if (exp.reached_time() > 0.99 × target_time):
        if (not started_valid):
            stop := 1
        fi
    else:
        if (started_valid):
            stop := 1
        fi
    fi
until(stop)
```

Figure 5.1: Pseudo-code implementation of the first part of the Cold Start Protocol: Rough localization of the validity limit.

Limiting the Boundaries of the validity limit: To get a more precise idea where the validity limit is – although looked at as an interval – we hope to limit ι_x and ι_y to an interval that is as small as possible. To do this, the procotol uses a determined bisection procedure at the exponent level. Since we know – with the reservation of random phenomena – that η_x is the 'valid' and η_y the 'invalid' exponent, we run another experiment with $\iota_z = \beta^{\eta_z}$ with η_z determined as follows:

$$\eta_z \quad = \quad \frac{\eta_x + \eta_y}{2} \tag{5.2}$$

If the experiment with ι_z is terminated with a valid outcome, we go on with $\iota_x = \iota_z$ and $\eta_x = \eta_z$, otherwise $\iota_y = \iota_z$ and $\eta_y = \eta_z$. Using this

```
eta_x := first_part.highest_valid_eta()
eta_y := first_part.lowest_invalid_eta()
beta := first_part.beta()
target_time := first_part.target_time()
for i:= 1 to 5 step 1:
    eta_z := (eta_x + eta_y) / 2
    iota := beta ^ eta_z
    exp := new experiment('xy', iota)
    exp.execute()
    if (exp.reached_time() > 0.99 × target_time):
        eta_x := eta_z
    else:
        eta_y := eta_z
    fi
rof
```

Figure 5.2: Pseudo-code implementation of the second part of the Cold Start Protocol: Narrowing the validity limit using an exponential bisection procedure.

bisection procedure, the range of the validity limit is narrowed repeatedly. The implementation of this part of the procedure is depicted in figure 5.2.

As the validity condition does not allow any conclusion about how close to the validity limit the experiment configuration is, it is difficult to generally conclude how much approximation is needed and how good the yet identified margin values are. For practical reasons, we limit this bisection procedure to be executed 5 times. We found that 5 iterations lead close enough to the validity limit value in the vast majority of cases.

5.2.2 Alternatives to Cold-Start

The Cold Start Protocol recognizably aims at locating a maximum within a small amount of steps. A mechanism implementing this procotol performs well, because it is capable of locating a local maximum value for typical models with practically no information about the solution within a small amout of steps.

Alternatives to the Cold Start Protocol may be based on mechanisms able to draw more information from the executed experiments to localize a maximum in even less steps. Using a combination of performance indicators, such mechanisms are supposed to select the next parameter for the workload

insertion variable as closely as possible to the maximum, such that the approximation is finally more efficient. We recommend this approach for localization strategies tailor-made for specific kinds of model.

5.3 Localization Strategies for Multi-Dimensional Workload Variation

5.3.1 Basic Concepts

Multi-dimensional localization is more complicated and needs more effort than one-dimensional localization. Before running multi-dimensional localization, users must reduce the evaluation complexity by simplifying their usage scenario whenever possible. If the quality of the evaluation results is not diminished afterwards, the dimensions for localization problems should be reduced to a minimum, since every additional dimension adds an order of magnitude for the time the evaluation process takes itself. The usage scenario has to be realistic and appropriate, and yet, in many cases, there is a positive correlation between different kinds of workload. For these kind of cases, the different kinds of workload can be expressed and encoded as probability based differenciations of one kind.

Even disjoint kinds of workload may have positive or negative mutual impact on the performance. A positive impact means that workload elements of a type W_1 do not hinder elements of a type W_2 as would if $W_1 = W_2$. It can be seen on a maximum throughput graph, if the graph has a convex form. Negative impact on the other hand is seen as a graph with concave form.

If a convex curve leaves the square delimited by the individual maxima for a configuration of zero other workload elements, this would mean that adding workload of a specific type W_2 improves the throughput for type W_1 and vice versa, such that W_1 with W_2 performs better than W_1 alone. However, we consider this as a special case, especially for disjoint kinds of workload.

5.3.2 A Modified Cold-Start Protocol

The modified Cold-Start is made to identify the maximum for each kind of workload assuming that no other workload elements are inserted. I.e., if there are n kinds of workload, the modified Cold-Start makes n localizations of maximum throughput, each time in a way that all other workload types have a zero workload insertion stream.

This strategy is required to determine the dimensions of the solution space. In general, we assume that the solution space in the n-dimensional problem space is the 2^{-n} segment of the n-dimensional sphere.

5.3.3 The Warm-Start Protocol

Every good localization strategy for the maximum margins is aimed to use a minimum amount of steps. The maximum margin is seen as the surface of the 2^{-n} spherical segment on the n-dimensional sphere. Such a surface is approximated best using $(n-1)$-dimensional circles normal to zero.

To do this approximation, we apply a bisection procedure on the level of n-dimensional angles. An angle means a fixed relationship between different dimensions; the variation is done with respect to the size of the workload streams, but a fixed relationship between the workload steams. I.e. for each configuration, a point on the straight line going through the origin point is selected, and the specific insertion rate per workload type is selected for the configuration accordingly. Variation thus means moving on that straight line.

Other than for Cold-Start, the approximation does not start at an arbitrary value; it rather uses the knowledge already present coming from the previously executed modified Cold-Start experiments and previous Warm-Start experiments, as far as available.

5.4 Variations of the Model

Localizing maximum throughput values is only one aspect of finding software solutions that perform well. During a software engineering process, where designers have to make decisions about system architectures and structures, more behavioural know-how is required.

As we have only looked at changing the experiment and finding out as much as possible for a given resource model, we now turn to looking at effects of changes applied to the model. These kind of techniques are especially helpful for designers who have to choose amongst different approaches or design alternatives.

The following three questions have been found most urgent for system architects:

1. *'How will a system behave if specific characteristics of one component are changed?'*
2. *'If some component is made redundant, what are the effects on the system's performance behaviour?'*
3. *'How does a system behave, if its structure is changed?'*

These three questions are addressed in this section, since they can all be implemented and investigated by modifications applied to the model subject to observation.

5.4.1 Changing Resource Characteristics

Changes in resource characteristics can happen daily in a software architect's life. There are many plausible reasons for such changes:

1. New and faster components are made available.
2. Programmers find better service implementations or code with better performance behaviour in situations with high load.
3. Other components are used in a different way with respect to load distribution or response time.

Performance evaluators have to respect these facts, since they have real influence on the individual component's runtime behaviour.

Another reason for changing resource characteristics is testing for realistic scenarios: There is room for improvement in every code and every system.

However, if improvement is necessary, it is better to improve at a point where considerable effects for the performance can be achieved for realistic situations. E.g., if we think of a system with three identified code sections, which might be candidates for code optimization, we'd like to know which of the modifications is most promising for improvements under realistic assumptions.

Resource characteristics variation is a powerful means of determining the effect of such changes.

Bringing the Variation into Effect

Before we can examine the effect of changes to resource characteristics, we have to consider what variation can be applied. If the change concerns a real code section including code statements and code structure, there are certainly only a few alternatives to choose from. In other words, there is a discrete set of possibilities, out of which the configuration of each experiment has to be determined. Variation consists of going through these possibilities and running experiments for the corresponding configurations.

The situation is different if there is a range of values, e.g. for numeric parameters. Theoretically, there are infinite possibilities of values in a continuous range, which is, of course, not a feasible base for evaluations. The following approaches for variations to handle a continuous area have been found useful:

1. The users declare that they want to explore the range to a specific resolution, which has to be indicated. Using this approach, a strategy can divide the range into pieces until the desired resolution is met. Each piece is represented by a specific value within the range, often the mean value. This approach is used to determine discrete values from a continuous domain space.

2. The users declare that the determination of values from within the range has to happen based on the identified results. This also gives the strategies discrete values, but result based and not domain based. An explanation of this approach will follow in the next section. This approach may be useful, if a local maximum of a key indicator is to be localized for a configuration within the range.

Application Examples

For the comparison of two or more implementations of a service, users will choose a variation strategy working with a discrete set of values, each value representing one possible implementation. Althouth this may appear to be trivial, this strategy is very useful in combination with other strategies. For example, if there are three of these variations, each for a different resource and each one independent of all others, the repeated application of this variation could produce the best solution in collaboration of the three components.

Sometimes users want to learn more about the influence of a specific parameter in general. To do that, they typically choose the parameter to vary within a range. To simply learn about the effect on the performance, they choose a variation strategy able to look at a range to a specific resolution. For example, think of a service implementation that uses two other services, s_1 and s_2, such that 30% of the calls go to s_1, the remaining ones to s_2. Assume further that s_1 and s_2 are exchangeable to a certain degree. A system designer is now interested to learn whether there are major performance differences, if the 30% parameter is varied in a range of 20% to 40% (case 1 from above). The strategy breaks the range into the desired pieces and runs an experiment for every configuration.

Instead of simply getting an impression, the designer may also choose to use a strategy that localizes the parameter with the maximum throughput. This strategy follows a result-driven parameter value selection (case 2 from above). Its algorithm contains an assessment function which allows conclusions on where to continue with the experiments. It continues its parameter selection and experimentation until the localization criterion is met. Using such a strategy, users do not get a good impression of the performance over the whole range of possible values, but they get the configuration, which led to the best performance. In addition, this strategy often needs less experiments and is thus faster.

Relationship with Other Strategies

Variations on resource characteristics are often combined with other variation strategies. In many cases, they use throughput localization strategies on a lower level of realization, since performance maxima may be sought for every configuration.

5.4.2 Increasing Redundancy

At some point of architectural design, system architects may want to find out whether they should introduce redundancy. This may be due to performance evalutations showing that their expectations were not met, or it may be for tactical reasons for a solution.

As distributed systems are often very complex, it is common for system architects, to look at the functionality and at non-functional aspects, such as distribution or performance, separately. The reasons why for the system architect should examine redundancy can be:

- To check with local redundancy whether a specific component is capable of performing better.
- To see whether distributed redundancy improves the performance behaviour at each location where the component will be.

The two cases mentioned above are very different with respect to the model setup, but they have collateral effects of variation in common that cannot simply be realized by increasing a performance parameter.

Bringing the Variation into Effect

To increase the redundancy locally, users have to increase the capacity[3] of the corresponding resource pool. The capacity change must be done carefully, since there may be collateral effects such as replication effort, group coordination and consistency ensuring protocols, and load attribution mechanisms.

Since every change in capacity may have considerable impact in the collaboration of the concerned resource pool with other resource pools, it is necessary to define the variation by means of constraints[4].

If the redundancy has to be introduced in a distributed way, such that the newly added components are situated at different locations, the variation is more complex. Such as for local redundancy, the realization for distributed redundancy has to be realized by means of constraints; however, every access to the component has to be re-visited:

[3]The capacity was introduced in definition 22 and discussed in section 4.2.4.
[4]Constraints were introduced and discussed in section 4.2.3.

- If the access was previously a call from a distant location, and if there is now a local component at that particular location, the call has to be re-formed, since the communication activities are now obsolete.
- If the access was remote and there is still no such component at the location of the caller, the system architect has to choose whether the call has to be directed to one of the locations with one of the new components, or whether there has to be a load attribution mechanism that routes the call. The latter is the most likely case with load balancing. It is very difficult to realize, though.
- To allow for consistent service provision, the service implementations have to be re-visited. In many cases, there are changes necessary in the service implementation as well as additional tasks to exchange information amongst the components.

The effects of distributed redundancy variation also have to be implemented using contraints. We suggest that there should not be too many possible configurations for the variation, since each configuration is already hard to realize in constraints.

Application Examples

Consider a client / server solution with a database as its central information repository. Introducing redundancy for the database either means to duplicate the database and run another database server centrally, or to install replica on other locations.

If the database is duplicated locally, a load attribution mechanism has to be introduced routing the queries to one of the databases. In addition, some of the calls will have to be changed, since they have to involve both databases in that the whole system must remain in a consistent state. Finally, some effort has to be added to keep the databases in-sync.

These aims can be realized in a new abstract component at the place in the model where the database previously had been, and by having the new component implement all the additional effort and synchronization mentioned above. Notice that we have described a case where the databases remain identical. If the tasks for the databases can be separated, this is considered as a design change and could be covered by variations described further in section 5.4.3.

For new databases instanciated at different locations there is no alternative to looking at each service using it and consequently adapting the service implementation. If a load attribution or even a load balancing algorithm has to be examined also, it is recommendably realized in an additional abstraction layer below the calling components.

Relationship with Other Strategies

Due to the complexity of these variations, we do not recommend using them in combination with other strategies, except for strategies that localize performance maxima.

5.4.3 Changing Model Structure

A variation in model structure is understood to be a variation in model design including the resource types, the resource pools and capacities, and the service implementations. Users typically want to vary model structure, to have performance related criteria for choices amongst design alternatives.

As the design of the model may change heavily during those variations, it is uncommon to make observations at the component level. In most cases, the measurements only refer to the overall key values.

Bringing the Variation into Effect

As explained for redundancy, model changes are complex variations. They can either be implemented using constraints[5], or by having the evaluation system compare two completely different models.

If the changes are applied using constraints, the users have to reflect the changes of the differing parts not only in the parts themselves, but also in the components using them. Each service implementation that generates workload for one of the parts (which are subject to changes) must be looked at, and the changes of the new services have to be reflected in the way the new services are called.

[5]Constraints were introduced and discussed in section 4.2.3.

However, our experience shows that for most cases, system architects prepare two totally different models, such that the modification only consists of exchanging the entire model.

A third approach of implementing structural change in some cases is the implementation in one single model using case differenciation. A model parameter serves at having the evaluation system decide which variation of service implementation and which internal workload generation is to be chosen for the current experiment. The application of this approach is very limited, though.

The strategy executing the variation has to ensure comparability. I.e., as each model is required to respond to exactly the same workload, the measurement must be made in a way that the key indicators of the different models can and will be compared with each other.

Application Examples

Some of the reasons why system architects want to evaluate structural changes are:

- Testing alternative distribution of logic. Assume that a piece of logic, e.g. a calculation or validation component, can be placed either on a server or on a client machine. For this case, most system architects are interested to learn whether there are performance gains by placing them on the client machines and whether the gains are high enough to justify the effort.
- Changing the call structure. Many modern servers are passive, reactive systems. A request is sent to them, they create a reply and send it back. As a consequence, clients that make specific requests regularly are forced to do polling[6]. The so-called push technology is an alternative. It uses the 'publish/subscribe' pattern[7], where the client that is to receive the information subscribes to a 'publishing' or 'pushing' service. The client will then be contacted by the server upon every change of information or periodically, as long as the subscription remains.

[6]Polling means that clients have to start a request for information themselves, to get the current information. An example of a polling based service is the so-called 'Simple Network Management Protocol' (SNMP), cf. [CFSD90] and [Ros96].

[7]Cf. [GHJV95] and [BMR+96] to learn more about active and passive interaction patterns.

Both the active and the passive implementation are supported by different middleware approaches. CORBA, e.g., supports the so-called object request broker[8], a passive or re-active element, and the so-called CORBA Event Service[9], an active element.

A variation of the call structure is a switch of the active and passive roles within a model. This means considerable effort to re-design the model and it is not feasible for every kind or workload.

As the variations typically concern major changes in model structure, there is typically a fixed number for configurations to be examined. Gradual variation is typically not part of the strategy.

Relationship with Other Strategies

As the compared models typically differ in large parts, it is very difficult to combine this strategy with other strategies. Every other strategy concerning the model would have to consider all configurations of the structural variation to implement their variation. In practice, this has not been considered feasible.

However, the workload variation strategies used to localize performance maxima, are typically required to compare the different model configurations.

5.5 Scalability

Scalability is a special performance issue since it looks at special questions and uses special variations on the model to be observed. It is important because finally, it is aimed at answering the question of how much performance can be reached ultimately with a system, assuming that the system's capacity can be increased without limits.

[8]Cf. [Gro02] for a specification of the Common Object Request Broker Architecture.
[9]Cf. [Gro01] for the specification of the CORBA Event Service.

5.5.1 Importance of Scalability

The importance of scalability comes from the ultimate consequence a system design decision may have. System builders or system buyers may prefer a system that scales well over one that scales less. Although, at the moment, the latter may be cheaper or easier to handle.

The idea is that either the better scaling system will turn out to be cheaper in the end, or it will achieve a level of performance that would never have been possible with the alternatives.

5.5.2 Basic Concepts

Evaluating the scalability of a model is not straightforward, as the capacity of a system has to be enhanced in controlled ways. This necessity arises from the observation that in most cases a simple enhancement of resources to a resource pool does not reflect reality. What we find more often is that such an enhancement changes the way workload is processed either by taking away bottlenecks and creating new ones in other places, or by making additional workload, such as synchronization or data replication effort, e.g., necessary.

A capacity enhancement is thus expected to cover these kind of collateral effects. The immediate question then is, how much more performance is possible after the enhancement? Ultimately, given the theoretical possibility of unlimited enhancements of the capacity, the question is: *'what is the absolute performance maximum of a given model?'*

Due to the stochastic nature of the models, double maximization is required to address scalability analysis. Double maximization refers to a maximum localization on the level of varying workload, driven by another maximum localization on the level of varying capacity. With no random phenomena, an analytical approximation might be to build a curve of experiment data and to extrapolate a maximum curve value.

5.5.3 Constraints for Capacity Variation

Realistic and plausible scalability analyses require the capacities to be changed in a realistic way and with all consequences. A change in capacity appearing without collateral effect is not the regular case. In many cases

there are collateral effects, such as additional internal workload needed by the concerned resources to offer their service consistently.

For example: Given a solution with a database system that performs well for the current usage situation. Upon change of requirements, such as increased number of users or additional jobs to be carried out by the database system, the workload exceeds the amount the database system can handle. To resolve that problem, another database system including hardware is added to the landscape. Unless the two database solutions do not offer services that are completely independent, the logical coupling between them has to be reflected. In the extreme case, complete replication and multisite transaction protocols have to be introduced, adding more effort to be carried out by each individual operating system.

The constraint that encodes such ways of capacity enhancements has to make sure that collateral effects of any kind are considered and that the systems after variation represent a realistic situation. Unrealistic situations must not be admitted by the constraint, or the analysis mechanism will produce wrong, misinterpretable results.

All previously analyzed states must also be included unchangedly. In fact, if introducing a possible capacity enhancement for a component that had previously been fixed with a fixed amount of administration effort etc., the constraint has to make sure that the previous configuration is still possible as one of the examinable configurations.

5.5.4 Maximum for One Kind of Workload

If there is only one kind of workload, the maximum localization corresponds a 'double Cold-Start'. One parameter to be varied is the capacity. The other one is the size of the workload stream. For each capacity configuration, a localization of maximum throughput is executed. Given defined possibilities of capacity enhancements, the 'absolute' performance maximum is the highest local performance maximum identified during the variation of capacity.

For our research, we have not implemented it as 'double Cold-Start' but rather as Cold-Start followed by a series of Warm-Starts. To do that, we have examined the performance maximum of one capacity configuration. After that, we have continued with some of the next capacity configurations, each time executing a Warm-Start, i.e. a maximum localization departing from the previously identified maximum value.

5.5.5 Maximum for Multiple Kinds of Workload

The same rules as for single-workload capacity enhancement analysis are also applicable for the case with multiple kinds of workload. Here, too, the localization of the throughput maximum is looked at as dependent of the variation of capacity. Notice that every additional independent kind of workload adds another order of complexity and another factor of analysis time. It is recommended to reduce the number of disjoint kinds of workload to a minimum.

5.5.6 The Simulation Quotient

Before looking at scalability issues, we have introduced several techniques to reduce the simulation effort and to accelerate the evaluation. When evaluating scalability, new questions arise from the necessity of the simulation model to reflect the policies of the target system, which are closely related to the fact that scalability studies deal with varying numbers of resources in a resource model.

Considering, what is done when simulating a target system, we recognize the following cornerstones:

1. A target system is a model or excerpt of what (out of the reality) is considered to be relevant for examination.
2. The evaluation model is a model of the target system reflecting only the components and behaviour relevant for the performance analysis.
3. The behaviour in the evaluation model is not functional, but it is required to resemble the real-world performance behaviour of the target system as closely as possible.
4. If administrative overhead, e.g. for data replication or server synchronization is performance relevant, it has to be built into workload processing either by making specific service implementations longer, or by introducing a novel type of workload inserted from external sources.
5. If dealing with a fixed model, the amount of resources and connections remains fixed for all examinations, except for structural variation. If, however, the number of nodes in the model has to be varied, e.g. to examine issues of scalability, novel forms of phenomena can appear that are related to the way the simulation deals with such objects.

There are phenomena in systems simulation that come from the simulation itself rather than from the modeled target system. These phenomena appear due to the fact that some behavioural patterns have to be implemented as part of the evaluation model even if they were not part of the target system.

Example

Consider the example of a service provided by a varying number of server nodes. In reality, there are many approaches to determine the node to which an individual client program has to address its requests. These approaches are relevant for the overall system's run-time behaviour, so they should be considered. To illustrate what can be done, we select the 'join-the-shortest-queue' policy that has the clients direct their requests to the server node with the shortest queue of pending requests.

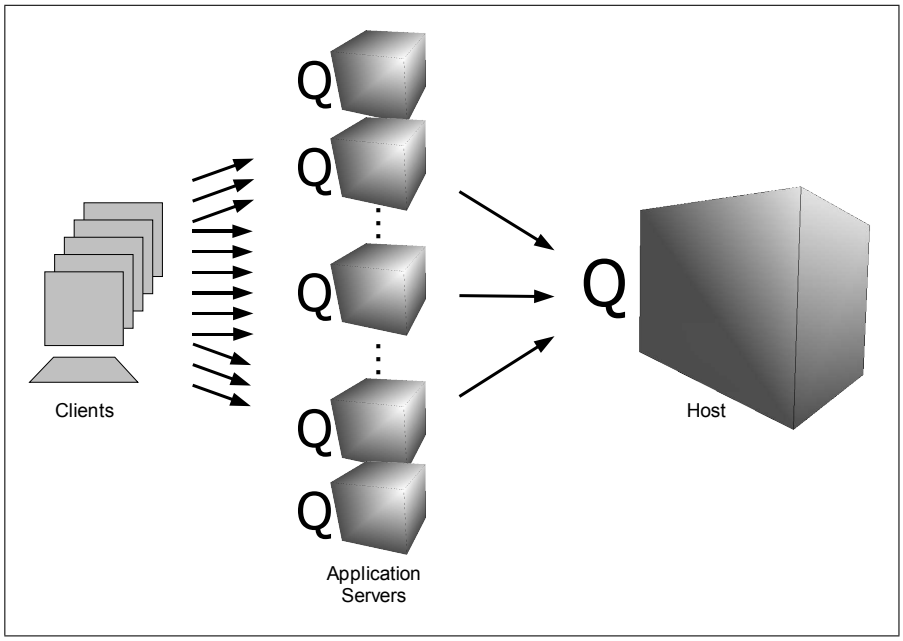

Figure 5.3: Example of a scenario for scalability analysis. Multiple clients send requests in 'join-the-shortest-queue' policy to one of the application servers, which in turn sends its requests to the host.

The search for the shortest queue has its cost in run-time. This should be reflected in the workload type's implementation in the model. However,

even if we neglect such kinds of run-time effects and only encode the 'join-the-shortest-queue' approach in the simulation model, i.e. as part of the simulation model, there is still observable influence on the simulation results.

Assume that there is a simulation procedure that determines the node with the shortest queue and that consumes no simulated time. Such a procedure comes at no cost of run-time for the simulated environment, i.e. executing it does not have the simulated time step any further. But, of course, it comes at some cost with respect to simulation time. The more nodes added to the evaluation model, the longer it takes to execute that procedure in the simulation system. Even if the simulated time does not go on, the simulation time, i.e. the time it takes to run the simulation, still goes on.

As a result, some part of the limited simulation time is 'eaten up' for executing the procedure that is presumed to be 'for free'. We look at the time taken to execute those procedures as time not available for the core part of the simulation. Thus for each simulation execution, there is a relationship for the core simulation activity time and the available / used simulation time.

Definition 25 *The relationship between the time used for core simulation activities and the overall time taken for the simulation (simulation time) for one experiment is called the* ***simulation quotient****.*

The simulation quotient is a key value users must be aware of when addressing complex evaluation questions, especially scalability questions.

Chapter 6

Evaluation System Technology

The construction of a performance evaluation tool must be built as much as possible on the base of recognized theory and conceptions. It was one of the general conditions of our research activities — to bring theory and conceptions together and build a well-tuned working unit. The same holds true for implementation issues. Instead of developing yet another monolithic application, we decided to make full use of all tools able to represent major aspects of our models and solve a part of our process in the best possible way.

Core tasks for our evaluation system are:

1. System modeling
2. System simulation
3. The application of strategies and instruments
4. The preparation of results.

Apart from these main tasks, there are auxiliary tasks to be covered. An example is making components collaborate, especially with respect to process control and data transfer.

In this chapter we look at the tools used to implement the performance evaluation system introduced in the previous chapters. In the first section

we look at HIT[1], a modeling and evaluation tool which covers the core tasks 1 and 2, modeling and simulation, largely.

Section 2 covers the implementation of control logic. It is realized as a framework based on workflow technology[2] and implemented using the Perl[3] programming language.

In section 3 we will take a look at some utilities used as helper applications, such as GnuPlot[4] used for presenting experiment results graphically. Finally, section 4 contains a generic architecture proposal for our approach. We show the connections between the different parts and present a system construction architecture for a new evaluation tool.

6.1 The HIT Evaluation Tool

HIT^{TM}[5], the *Hierarchic Evaluation Tool*, was developed at the University of Dortmund, Germany. It consists of several parts that connect modeling and evaluation in a handsome way.

HIT essentially consists of the following three parts:

1. The *HI-SLANG*[6] language, used to define (resource) models as well as experiments,

2. the evaluation engine including an analytical part for simple models and a simulation part[7] used to run more complex experiments, and

3. the graphic user interface *Hitgraphic*^{TM}[8] to build models graphically. It facilitates inserting program code at the right place and running individual simulation experiments, directly from the modeling tool.

[1] HIT is the HIerarchic evaluation Tool, © University of Dortmund, Germany.
[2] Cf. [vdAvH02] for an introduction to workflow management.
[3] `http://lang.perl.org`
[4] More on GnuPlot can be found on its website `http://www.gnuplot.info/`.
[5] Cf. [BMW93] for a short description of HIT.
[6] Cf. [HIS93] for the reference of HI-SLANG.
[7] The simulation part of HIT is based on Simula.
[8] Cf. [Hit93] for more information on Hitgraphic.

6.1.1 Characteristics of HIT

Modeling Characteristics

HIT's experimentation is based on artifacts that allow building system models quite similar to those introduced in chapter 2. This is not a surprise, but it shows how many good constructs and conceptions of HIT have been included in our approach.

HIT's biggest advantage is its hierarchic approach for modeling and simulating resources. It is especially easy to configure resource properties, such as the way of load processing[9], the queueing discipline[10], or processing speed.

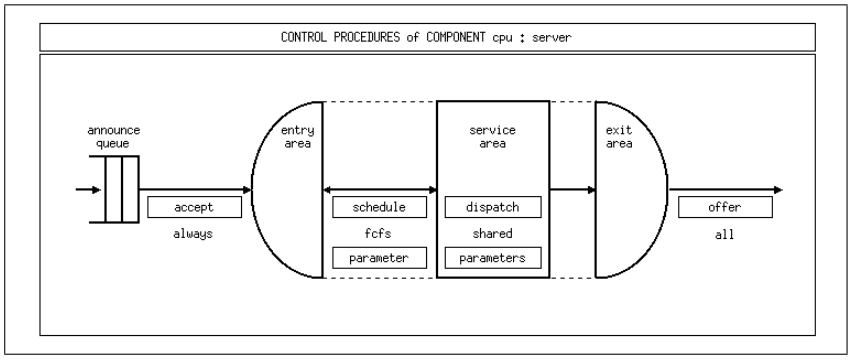

Figure 6.1: The definition of resource properties in HIT using Hitgraphic's resource properties dialog.

Another advantage is its support for graphic representation and easy program coding of services. The services to be offered by a resource are represented clearly, references to lower level services are easily made, and implementations are done in a few lines of *HI-SLANG* code[11].

But:

- HIT has no conception of resource pools. Instead, users can specify special attributes regarding execution mode and speed in a way that most aspects of resource pools can be reflected and included.

[9] Either in parallel or sequentially.

[10] Cf. definition 11 in section 2.2.2.

[11] This includes programming of quite complex constructs, such as workload generation, the use of probability distributions, loops, and conditional branching.

- Resource types are called 'components' in HIT. They cannot be reused the same way as types in our formal model, as parametrization is not possible.
- There is no possibility to define the degrees of freedom for model or experiment variation.
- Especially there is no possibility to model structural variations or to define the necessary constraints.
- There is no support for strategies as introduced in chapters 4 and 5.
- HIT's capacity term is different from what we understand capacity should reflect.

However Hitgraphic is not only very comfortable to use but also offers features that support fast-cycle model development. That includes fast model consistency checking, embedded code syntax checking, as well as full HI-SLANG code generation and execution.

Characteristics of Experiment Execution

To execute an experiment means to prepare both the model and the corresponding experiment for the specific case, then to run the simulation, and finally to identify and process the result values. An experiment including the resource model can either be set up in Hitgraphic with code generation, or it can be programmed in HI-SLANG code directly. Such a HI-SLANG experiment can easily be ran by calling HIT from the command line or from the Hitgraphic tool.

Experiment execution in HIT consists of the following steps:

1. The generation of HI-SLANG code, if the model was set up graphically
2. The generation of a Simula program from the HI-SLANG code
3. The compiling and linking of the simula program[12] into a binary stand-alone program

[12]In the case of the HIT license granted to the University of Zurich, HIT uses the so-called CIM compiler that translates Simula code into C code; the latter is then compiled and linked into a stand-alone binary program using a standard C compiler.

4. The execution of the stand-alone program, which eventually produces the previously defined reports as its output

As mentioned above, the conceptions of HIT do not cover all our requirements. Variation of any form is not supported. The following aspects are missing:

1. HIT allows for series of experiments to be ran if the values to be varied can be expressed as HIT parameters, and if the discrete values to be used for the parameter are pre-defined. Unfortunately, this excludes every kind of self-adaptive variation, for which the results of a previous experiments are used to determine the parameters of the following ones.
2. Variation of the model is not part of HIT's conception, HIT models are static. Its experiment series mechanism is not applicable for the kinds of model variation required for our approaches.
3. Due to lack of variation, there is neither an appropriate notion of results presentation.

For this reason, we have decided to use HIT to a degree that allows external control and to implement the remaining functionality ourselves. We take Hitgraphic as an easy-to-use modeling and code generation tool and HIT's simulation engine to run individual experiments. However, the realization of model variation and strategies are implemented as complementary steps; they produce the experiment specifications that are required for corresponding evaluations. This part is described in section 6.2.

Characteristics for Realizing Strategies

As mentioned above, major strategies of our approach are not supported by HIT. On the other hand, we wanted to use HIT's expressive modeling and experiment language and its powerful simulation engine. Several features contribute to an implementation of the strategies defined in chapter 4:

1. The preparation of an experiment including the resource model, all in the form of HI-SLANG code, especially:
    - Changing the workload insertion rate in the HI-SLANG code

- Changing parameters in HI-SLANG code
- Changing the implementation of internal workload generation
- Changing probability distributions in the HI-SLANG code
- Changing model structures in the HI-SLANG code

2. The process control for HIT's simulation engine
3. The result identification and extraction
4. The determination of the next step in the strategy

By supporting interaction with the experimentation and simulation engine in these ways HIT offers great flexibility and has shown to be very useful for our strategy implementations. Although the implementations have to be done outside of HIT's mechanisms we can still draw full advantage of its features and its conceptions.

Characteristics for Result Preparation

As there is no real support for our strategies in HIT, there is no mechanism supporting our requirements for result preparation either. However, there is result preparation for individual experiments or for HIT-defined experiment series.

There are several forms of output offered by HIT, all more or less reports in text format. For our result preparation and presentation we used the text output feature by having HIT make tabular output for every experiment and filtering out the key values with an appropriate tool. By having HIT generate this output in the same way for each experiment of an evaluation series, we eventually have different series of indicator values related with each other by their quality over the whole evaluation series.

By connecting this set of information with the set of experiment parameters used to run the series, we can prepare value series that show the connection between the applied variations and the corresponding results. This is required for the interpretation of the variation effects.

If structural changes to the model eliminate measurement points, it is sometimes hard to build appropriate series of <`parameter,measurement`> pairs. For this case, we have typically based our investigations on the available data on the level of the entire model and not on measurements for particular resources.

```
HIT  Version 3.7.004    TABLE    2004-07-30  15:48  PAGE  1    University of Dortmund
Control                 File expone.hit
Experiment Name         expone
Model Type              dbpagelocal
Model Name              evalone
Model Parameters        Name              Type          Actual Value
                        ============================================
                        seed              INTEGER                13
Used Method             SIMULATIVE
Date of Compile         2004-07-30        Time of Compile     15:44
Start Date of Run       2004-07-30        Start Time of Run   15:44

Stop  Date of Run       2004-07-30        Stop  Time of Run   15:48
Used CPU Time           177.40000
Reached Model Time      238.47500
*******************************************************************************
HIT  Version 3.7.004    TABLE    2004-07-30  15:48  PAGE  2    University of Dortmund
Evaluationobjectname : evaloneobject
-------------------------------------

Hierarchy   |Esti |         THROUGHPUT          |
============+=====+=============================+
ALL         |Mean |      1.153161               |
            +-----+-----------------------------+
            |Stdev|      0.784582               |
            +-----+-----------------------------+
```

Figure 6.2: Excerpt of one of HIT's generated result files.

Characteristics for Result Presentation

HIT produces key indicator results, but there is no support for expressive graphical result presentation. Although, as mentioned above, there is some support for running experiment series in HIT, for which HIT would produce the output in a combined result file. This feature is not helpful or applicable for the implementation of the strategies presented in chapter 4.

Our understanding of a suitable result presentation is a representation in a way, which avails conclusions and the recognition and interpretation of interrelations. We found graphical representation the most useful form of presenting results. The measured key values are represented in a graph as function values for the varying configuration parameters. If not applicable, table based representations or other forms of diagrams can also be helpful.

6.1.2 The Practical Use of HIT

As our research included the experimental usage of standard tools, we refrained from building a fully integrated environment ourselves. Instead,

Insertion Rate	Throughput	Response Time	Population
0.5	0.512886	0.159387	0.081757
1.0	1.012469	0.163031	0.165043
1.5	1.508204	0.162940	0.245837
2.0	2.002240	0.161092	0.322643
2.5	2.546429	0.161193	0.410412
3.0	2.973135	0.150231	0.446599
5.0	4.957761	0.160232	0.794701
7.0	6.985238	0.168415	1.178185
9.0	9.222918	0.160474	1.479832

Figure 6.3: Example of a result series compiled of information of HIT result files.

we used some excellent parts of HIT and implemented our strategies outside on top of HIT. As mentioned in section 6.1.1, the used parts of HIT include Hitgraphic for model building and HI-SLANG code generation, and HIT's simulation engine to run individual experiments and to determine key indicators.

We have implemented a framework[13] to realize our strategies. The framework uses the mentioned parts of HIT by modifying the Hitgraphic generated HI-SLANG model and having HIT's simulation engine run the experiments, while process control and strategy realization is implemented in the framework.

In the day-to-day usage HIT has proved to be very reliable and stable. The only problem appearing from time to time was the unstable Solaris^{TM}operating system that caused the machine to boot from time to time. We think that the reason for this effect is the exhaustive processing during HIT's simulation experiments.

6.2 Evaluation Framework in Perl

In chapters 4 and 5 we have presented some strategies to implement our approach of performance evaluation. Since, as of today, there is no tool to

[13]The framework is described in section 6.2.

implement these strategies, especially in combination with other tools for simulation, we have realized a tool ourselves. It uses other specialized tools and integrates them to a whole. Since our tool plays the role of a frame for the specialized tools, we have considered and implemented it as a framework that uses the other tools' features without in-depth knowledge of them.

6.2.1 Experiment Execution

From an integrated tool's perspective, experiment execution consists of two tasks:

1. Process control and error management
2. Collecting and saving the results

Process control first consists of starting a process with a HIT simulation task including specifying the correct path, environment variables, and the prepared HI-SLANG experiment file. As simulation is typically a strongly computational task that may cause crashes for average machines, the surveillance of undesired process terminations, error codes returned by the process, and process output is also part of process control.

HIT reports the result of each experiment as a text file. This file has to be saved in a way that the control and configuration data for that particular experiment is connected with the result file. In addition, the chosen strategies and their discrete experiment states, i.e. the experiment configurations and control data series, also have to be stored. Storing these kinds of information allows to re-use evaluation results already found in any context.

A nice little feature of our framework is the built-in wait function: Since simulations may run with high process priority, other tasks of the used machines may be postponed by the machine's operating system. To allow the operating system to handle its low priority tasks, e.g. to restructure its virtual memory pages, we have added a wait statement after each simulation step.

6.2.2 Implementation of the Strategies

The presented strategies have different requirements for experiment execution and flow of control and configuration data. Some strategies need to be executed in specific ways and in specific order, e.g. to accomplish self-adaptiveness. This level of control is a kind of meta-strategy, which has not been subject to further research.

The question, what strategy should be applied at what time, is not addressed at the moment, neither by our tool nor by the users. As of today, the users choose a strategy by starting a corresponding procedure[14]. The strategy is then followed until it complies with its termination criteria.

Every strategy consists of a series of experiments. For example the Cold Start Protocol[15], e.g., determines the validity limit of a system's performance by varying the workload insertion rate based on the results of previous experiments. Such mechanisms require the availability of results of previous experiments of the strategy's series, as well as corresponding configuration data at execution time.

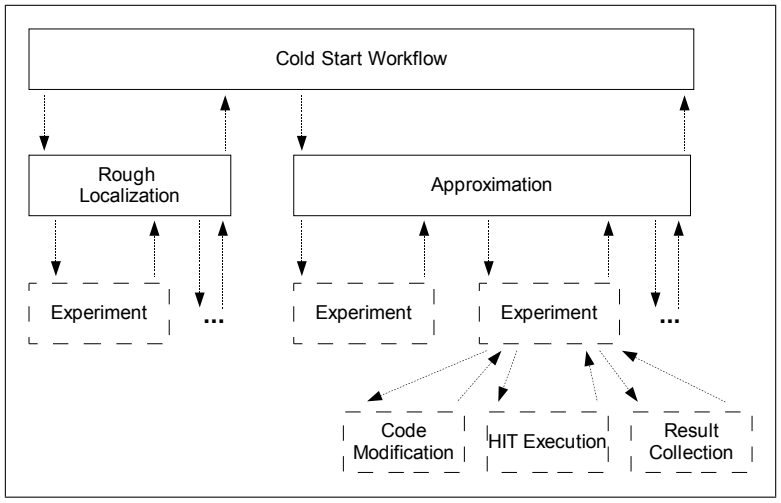

Figure 6.4: An example of an actual workflow showing its object-oriented hierarchic structure. The solid boxes are steps defined to start the workflow whild the dashed-line boxes are instanciated at run-time according to the actual situation.

[14]To the framework tool, the procedures are available in the form of standardized workflow definitions.

[15]The Cold Start Protocol was introduced in section 5.2.1.

6.2.3 Code Preparation

In our approach, we run experiments using HIT's simulation engine which requires HI-SLANG source code. To be able to use the excellent conceptions and simulation capabilities of HIT, and to nevertheless realize the presented strategies, our framework has to control HIT's models and experiments. The straightforward way for us was to realize these capabilities on the level of HI-SLANG code.

Code preparation consists of several steps:

1. An *original experiment* must be made available in he form of HI-SLANG program code, and
2. a configured HI-SLANG coded experiment must be prepared for each simulation experiment using code modification techniques.

Preparation of an Original Experiment: HIT offers an intuitive way of building resource models with its Hitgraphic graphical user interface tool. Resource modeling includes implementing the services and defining the values to be measured at specific measurement points. In addition, there is a feasible way of defining an experiment and even running it out of the Hitgraphic user interface. However, as such experiments are outside the scope of our evaluation and experiment control, those experiments do not contribute to strategy realization.

Hitgraphic is able to generate a HI-SLANG representation of the models and code sections stored in its internal databases. This file will be passed to the evaluation tool as *original experiment* file. We could now easily run the experiment from the command line by calling HIT with the file name as parameter; that is exactly what Hitgraphic does for its experiment executions.

Our approach is aimed to make sure that experiment preparations and executions are possible from the evaluation control part. The final step in preparing the original experiment is thus to make sure that the experiment file is syntactically correct. This is done by issuing a compilation of the code file without later execution.

Code Modification: An individual experiment is prepared by placing the right configuration information, which is determined by the characteristics of a particular state in the evaluation series, at the right place of the HI-SLANG file. To do that, we have our framework read the original experiment file and modify the corresponding sections. The implementation relies on textual replacements using Perl's regular expression and textual pattern matching facilities. The modified file is then stored as experiment code file. To make sure that the right sections are replaced by the right values, the text replacement mechanisms have to be tested.

```
%TITLE MODEL/COMPONENT TYPE dbpagelocal
   COMPONENT disk : server (
      LET  ACCEPT    :=  ALWAYS,   LET  SCHEDULE  :=  FCFS,
      LET  DISPATCH  :=  EQUAL,    LET  OFFER     :=  ALL );
   COMPONENT cpu : server (
      LET  ACCEPT    :=  ALWAYS,   LET  SCHEDULE  :=  IMMEDIATE,
      LET  DISPATCH  :=  SHARED,   LET  OFFER     :=  ALL );
   REFER dbpage TO disk, cpu
   EQUATING
      dbpage.present WITH cpu.request;   dbpage.load WITH disk.request;
      dbpage.locate WITH disk.request;   dbpage.compile WITH cpu.request;
   END REFER;  {of component/model type dbpagelocal}
%TITLE ACTIVITIES OF MODEL TYPE dbpagelocal
   BEGIN
      CREATE 1 PROCESS dbpage EVERY negexp(10);
;
```

```
%TITLE MODEL/COMPONENT TYPE dbpagelocal
   COMPONENT disk : server (
      LET  ACCEPT    :=  ALWAYS,   LET  SCHEDULE  :=  FCFS,
      LET  DISPATCH  :=  EQUAL,    LET  OFFER     :=  ALL );
   COMPONENT cpu : server (
      LET  ACCEPT    :=  ALWAYS,   LET  SCHEDULE  :=  IMMEDIATE,
      LET  DISPATCH  :=  SHARED,   LET  OFFER     :=  ALL );
   REFER dbpage TO disk, cpu
   EQUATING
      dbpage.present WITH cpu.request;   dbpage.load WITH disk.request;
      dbpage.locate WITH disk.request;   dbpage.compile WITH cpu.request;
   END REFER;  {of component/model type dbpagelocal}
%TITLE ACTIVITIES OF MODEL TYPE dbpagelocal
   BEGIN
      CREATE 1 PROCESS dbpage EVERY negexp(100);
;
```

Figure 6.5: Excerpt of a HI-SLANG code example before and after code modification (from 10 elements per second to 100 elements per second mean rate, bold typeface). The modification is very small in this case but it may include entire sections in other cases.

As of today, all code preparation activities are programmed directly as Perl framework code. As such, they are closely related to the original file.

There are three possibilities for replacements:

- For code sections which are matchable without ambiguity, the concerned piece of code can be replaced in one Perl statement.
- For code section which are ambiguous themselves, but which are embedded in a larger section of code without ambiguity, the replacement is realized in three steps: In a first step we recognize the large section and store it into a Perl variable, in the second step the specific code part within the variable is recognized and the desired replacement is applied. The third step then re-matches the original large section and replaces it with the modified one.
- If there is a chance that the modifications may still go wrong, the original experiment file is changed and unique textual identifiers are inserted to facilitate unique matching. After that, the original file cannot be ran as experiment anymore, but code modification is guaranteed.
- If the modification concerns the replacement of very large parts, i.e. if the variations differ largely, the original file is better split into sections and the part that is subject to modification is implemented by code generation. The code modification mechanism is then required to assemble the sections and the generated parts to a new experiment file.

In any case, the result of code preparation must be a correct HI-SLANG script representing model and experiment for one specific case of a strategy, i.e. for one configuration.

6.2.4 Preparation and Presentation of Results

As mentioned in section 6.1.1, each experiment execution produces an output file containing the determined key values. The values are logically connected with the specific control and configuration data of that particular experiment. Result preparation is done by extracting the key values from HIT's report files and bringing them into connection with the experiment meta data.

The strategies, as the driving elements of evaluation, require the information about each configuration and the identified results to be stored and made

retrievable for later steps. Amongst the result data, the validity criterion introduced in section 4.2.2 is important information required by many strategy implementations. Other stored information includes the specific HI-SLANG source file, the Hit made results table, and internal decision data.

A specialized software component is aimed to retrieve the results from the Hit made report files. The component is often tailor-made since it must be able to handle specific situations correctly. For example, if model modification leads to the elimination of a specific measurement point, the component is aimed not to fail, but to handle that situation correctly. Finally, it brings the information into an order that allows users to interprete the completed strategies and the results found there. There are different useful orders, including the temporal ordering according to actual strategy execution, or natural ordering of configuration parameters.

After collecting the results, our evaluation tool is aimed to present a graphical representation or a report to the user, or to allow further statistic processing. For all reporting forms, the identified data have to be presented in specific forms. For example, if the results are to be presented as functional graph, depicting configuration data on the x-axis and results on the y-axis, the reporting output is aimed to be a GnuPlot data file; the generated file can then be processed by a corresponding GnuPlot script. Other reports require other forms of tables, including CSV[16] files.

6.2.5 Framework Structure

As mentioned earlier, we have implemented the framework in Perl[17] based on principles of workflow technology[18].

The ideas taken from the workflow standard are:

- Persistence of a possibly long-lasting 'procedure'
- Some elements of workflow structure and flow logic

[16] A CSV file is a comma-separated-values file containing individual values of a table separated by a comma or semicolon, with each record of the table on a new line.

[17] More on the Perl programming language and its inventors can be found on `http://lang.perl.org`.

[18] The standard for workflow management systems can be found on `http://www.wfmc.org`.

```
1               1
5               5
25              25
55.90169944     55.90169944
58.78517609     58.78517609
59.52899171     59.52899171
59.71641155     59.71641155
59.76335864     59.76335864
59.76482633     59.76482633
59.76556019     59.76556019
59.76574365     59.76574365
59.76576659     59.76576659
59.76576802     0
59.76576945     0
59.76577232     0
59.76577805     0
59.76578952     0
59.76583539     0
59.76592712     0
```

```
set terminal png small color
set output "throughput.png"
set size 5/5., 3/3.
set format xy "%g"
set logscale x
set nologscale y
set xrange [0.8:150]
set yrange [0:70]
set xlabel "workload elements"
set key below Left spacing 3 box
plot "throughput.dat" with linespoints
```

Figure 6.6: Left: Example of a data file made for GnuPlot; the data in the left column will be used as x-axis coordinates, the others as y-axis coordinates. Right: GnuPlot script to generate a corresponding graphics file.

- The capability to re-start a workflow that has been interrupted
- Information passing between steps
- The possibility to monitor the state of execution from outside

Other principles for setting up the framework were:

1. Hierarchic, object-oriented structure with a *workflow* as object on the uppermost level and other, more specific chain objects on lower levels. The structure can be of arbitrary depth.
2. Dynamic composition and de-composition of the object structure allowing for run-time determination of activity chains.

The Evaluation Workflow Structure

The evaluation is realized as the uppermost element of the workflow structure. Its characteristic as a meta-strategy dictates that it selects and

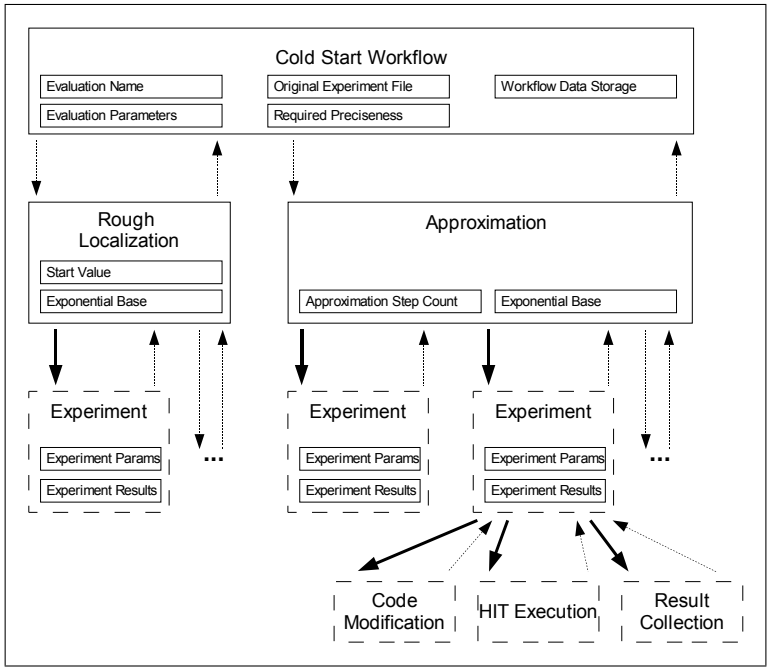

Figure 6.7: An example of an actual workflow. The big boxes denote the steps with the workflow root on the top. The little boxes within show some of the steps' state. The arrows symbolize the flow of control, the bold arrows also mean instanciation.

inserts strategy objects dynamically. Selected strategies are inserted as workflow objects in the level below the evaluation.

As we have mostly used strategy selection as the expert's activity, our evaluations have mostly been composed manually, often starting with only one strategy to be followed. As a consequence, strategy objects below the meta-strategy are added during the course of an evaluation, and the strategy series grew during execution.

Every strategy is run as a series of experiments. Each experiment is defined by a specific experiment configuration, which is determined by the strategy object. As many strategies rely on results of both own and other strategies' experiments, the series of experiments is not determined a priori. It is rather constructed during the course of the evaluation relying on the built-in mechanisms of self-adaption.

Below the strategy objects are thus the experiment objects, holding the necessary configuration data and controlling the execution of the experiment.

To run its activities, the experiment objects typically have some task-specific objects, such as the code preparation object, the HIT execution control object and the result collection object. However, if the chosen strategy relies on another strategy, like, e.g., the two-dimensional localization that relies on one-dimensional localization, the experiment objects are in fact strategy objects. In this case, the upper-level strategy object determines the fixed configuration data for each lower-level strategy object. The latter is responsible for determining the remaining fixed configuration information and controlling execution as described above.

A result collection and presentation object is often appended to each strategy object. This is plausible as each strategy produces results that are best seen and interpreted from the perspective of the strategy's scope. In other words, it is useful for users to look at the results in the light of what has been investigated.

In figure 6.7 we have depicted an actual workflow showing not only the execution order, but also the structure of a real workflow as described in this section. For a starting evaluation, the workflow is relatively small, while for an evaluation, which has executed some strategies, the number of workload objects is relatively big.

6.3 Tools and Utilities

A considerable amount of tools were used to implement the evaluation tool described in this work. The following are the tools we do not look at any further:

- Command tools from the C-Shell (`csh`) to Tcl / Tk
- Compilers, especially the Gnu C compiler[19],
- The CIM Simula-to-C compiler[20],

[19] More on `gcc` can be found on `http://www.gnu.org`.

[20] For more on the free CIM Simula, please refer to the following web-pages:
`http://www.volny.cz/petr-novak/cim/`,
`http://www.ifi.uio.no/~cim/sim_cim.shtml`, or
`http://www.gnu.org/software/cim/cim.html`.

- Additional development and debugging tools, e.g. for Perl[21],
- Database layer tools, such as the Gnu DBM Database Libraries GDBM[22].

The most important tools apart from HIT are those used to show the identified results visually or to further analyze them statistically. Further statistical analysis was not looked at as part of this work; however there is plenty of data from the simulations that could be examined. In this section we discuss result presentation. We look at the visual presentation as an example, since it is part of our work to show in what way identified results can be presented to facilitate correct interpretation by users.

During our studies we have created large numbers of diagrams and graphs. Typically, the most useful ones were two-dimensional functional graphs. We used them to depict result values in the y-dimension as function values of configuration values in the x-dimension. In other words, we depicted the configuration value of interest as the function variable on the x-axis, and for each chose value on the x-axis, the identified result value as function value on the y-axis.

Most of the graphical representations have been built with GnuPlot[23], a plotting tool able to create comprehensive 2-dimensional and 3-dimensional graphs with high flexibility. Amongst the advantages of GnuPlot is the fact that it chooses the area of the depicted dimensions by itself. It analyzes the values to be displayed and chooses a range that shows the area entirely. Another big advantage is its ability to depict the graphs using linear and logarithmic scales. If applied carefully, this technique allows to dive into the zones of interest more impressively.

A drawback of automatic range determination is that even the slightest stochastic effects may be depicted as large deviations. For example, if a result remains more or less equal over a whole evaluation, deviating only due to stochastic effects, GnuPlot tends to compose a graphical representation that shows the deviations as major differences in function values. This may be very irritating, especially for users with little practice in statistics.

[21] More on the Perl programming language can be found on `http://lang.perl.org`. For development with Perl, cf. the resources of ActiveState Corp. (`www.activestate.com`).

[22] For more on GDBM cf. `http://www.gnu.org/software/gdbm/gdbm.html` or `http://web.mit.edu/gnu/doc/html/gdbm_toc.html`.

[23] More on GnuPlot can be found on `http://www.gnu.org`.

6.4 Generic Architecture

To present and illustrate a good architecture for an evaluation also, we start from the process defined in chapter 3. The choice of a system architecture can be motivated by different factors. For our case, product characteristics and product maturity of the evaluation tool were much less important than conceptions and specific features. We were neither interested in questions of software distribution nor collaboration. But it was important to use the right tool parts for the individual tasks. Our architecture is thus expected to enable, facilitate, and control the usage of other, specialized tools.

The generic architecture presented in this section is thus to make sure that the main aspects of the evaluation process are covered, and to clearly show the logic parts such a system must contain. All special issues of programming approaches or component technologies are ignored, as are specialties of data persistence, except for the one needed by used tools, e.g. HIT.

6.4.1 Covering the Evaluation

The parts identified in the previous chapters all make contributions to the information required by system engineers. Their contribution can be represented in a simplified value chain format. The value chain we identified for our evaluation process consists roughly of the following four members:

1. The model specification
2. The definition of configurations for the evaluation
3. The evaluation itself
4. The presentation of the identified results

These process members are depicted in figure 6.8. All elements of the cyclic representation in figure 3.1 in chapter 3 can be attributed to one of the four members. Notice that these members are different from the core tasks enumerated at the beginning of this chapter. This is due to the fact that tools or components are not necessarily symmetric with the tasks of the evaluation process.

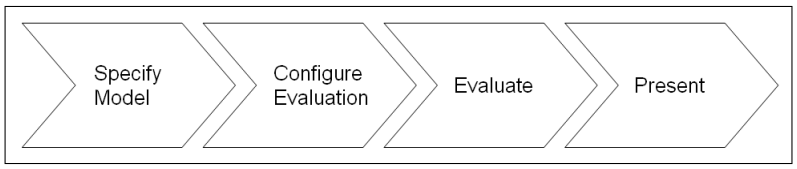

Figure 6.8: Schema of a value chain consisting of four member steps. The members are like categories, to which all parts of other process representations can be sorted into.

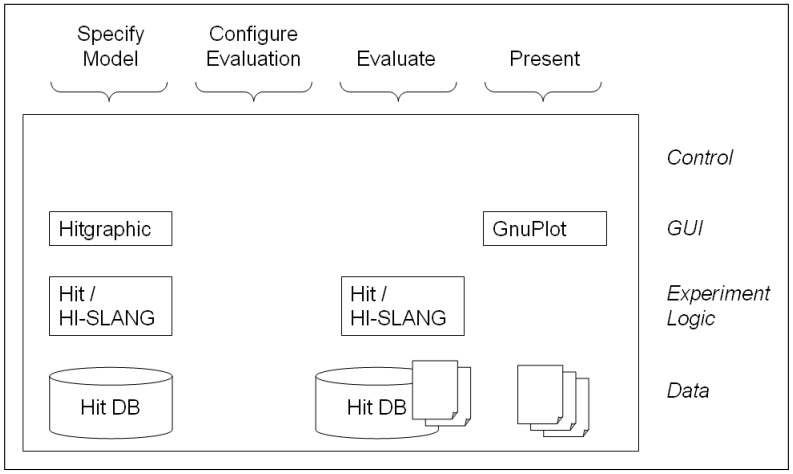

Figure 6.9: Schema of tools and parts identified earlier in this thesis and ordered according to the evaluation value chain. See figure 6.10 for the more complete schema.

In figure 6.9 we have depicted a tool landscape following the order of the value chain on the x-axis and showing the tools identified earlier in this thesis. Each tool addresses one aspect of evaluation in a specialized way. The y-axis is roughly ordered according to the level of contribution each tool offers for the realization of the evaluation process or its strategies.

6.4.2 Dynamics of the Process

The evaluation process is not implemented by simply connecting the tools in an appropriate way. As mentioned previously in this chapter, the strategies introduced in chapters 4 and 5 are not covered by the used tools. Necessarily,

these aspects have to be implemented separately, in tools or components yet to be identified.

Evaluations are aimed to produce information about the performance behaviour of a target system by applying hypothesis driven strategies. As most of the strategies behave in a self-adaptive way, i.e. they select steps based on the current state, the process is looked at as a dynamic, state-based approach. The applied tools however are mostly considered as static, often state-less parts: They expect some information as input, process it, produce some other information and quit.

Not all of the static parts are state-less, but since they have no notion of the process' way of collecting information, a connecting and steering control logic is inevitable for the process implementation. To find all missing parts of the architecture, we have to identify the static parts and find the gaps to be filled with new, dynamic parts.

The following static parts have been used in our implementation:

- Model construction by means of the graphical user interface Hitgraphic
- The representation of the model as HIT model or HIT experiment in the form of HI-SLANG code
- The execution of an individual experiment using HIT
- The functional representation of some identified results using GnuPlot
- Storing individual data and individual experiment configurations on the level of the HIT database or the HIT report files

The dynamic parts have to cover the white spots in figure 6.9. By implementing them, we are able to fully cover the contributions of each member step of the value chain. We have identified the following areas where some parts are missing:

- The complete layer of evaluation control
- A part to configure evaluations including strategy selection and constraints definition
- The control of the evaluation including the strategies implementation and the experiment control
- The extraction and preparation of data for graphic representation or other kinds of reports

6.4.3 Covering the Missing Parts

The areas that were white in figure 6.9 or that were identified as means of self-adaption have to be covered by parts of our process implementation framework. The parts are required to cover the whole evaluation process as described in chapter 3. They shall be introduced and explained subsequently.

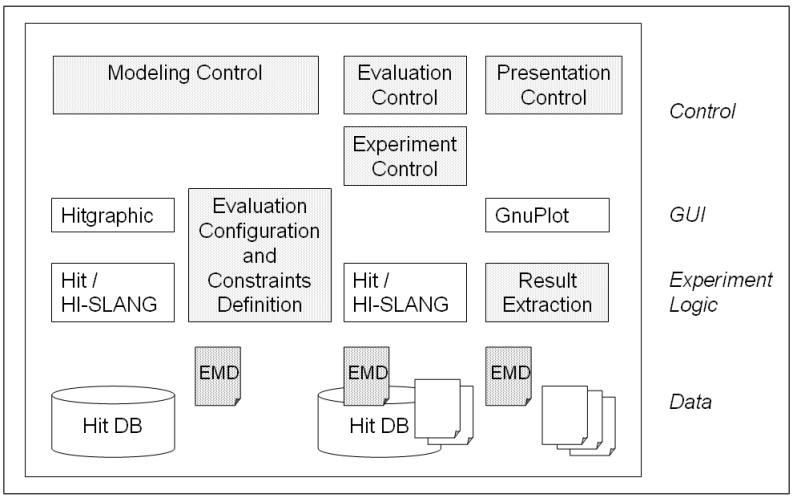

Figure 6.10: Schema of the used tools now enhanced with the missing parts to be realized by a performance evaluation tool (Figure 6.9 completed). This illustration also includes the Evaluation Meta Data (EMD).

There are parts for different scopes and aims:

1. To implement aspects of the process or its parts that cannot be covered by the used tools
2. To integrate the tools into a whole solution with respect to both control flow and data flow

The parts added in figure 6.10 have the following aims:

Modeling Control is required to control all modeling activities and to maintain the connection between resource (and service) models and the corresponding evaluation. In other words, it is aimed at keeping the system model and the evaluation choices in sync. This includes constraint definition and strategy selection.

Evaluation Control is required to control the progress of evaluation. This includes following and realizing specific strategies, having a specific strategy configuration executed. It also determines following an overall strategy which controls the order of strategy application. In other words, if there are different strategies, an overall strategy decides how to apply which one. The strategies are then applied, i.e. followed, and specific strategy configurations are instanciated. Each configuration leads to an experiment, for which the control is passed to the Experiment Control.

Experiment Control is responsible for the control of the HIT experiment engine to run an experiment for a specific configuration. This includes providing the given configuration's experiment code as described in section 6.2.3.

Presentation Control is responsible for having the evaluation tool present the identified results in a comprehensible way. This includes controlling the tools used for visualization, especially GnuPlot.

Evaluation Configuration and Constraints Definition This part allows for users to define the directions the evaluations should go. This includes the declarations of what can be varied and how. And it also includes the definition of complex constraints, i.e. limitations of which configurations of variations are admitted[24].

Result Extraction This tool prepares the results stored in individual experiment report files and brings them into plausible connection with the configuration information.

Evaluation Meta Data The meta data are used as persistent control data for the evaluation process. These data are required for the strategies, which use earlier results for determining their next step and configuration.

[24]Cf. section 4.2.3 for more on constraints.

Chapter 7

Results and Outlook

It was the aim of our research and of this thesis, to provide a methodology for the evaluation of distributed systems with respect to their performance behaviour. Different aspects of performance studies had to be analyzed, addressed by the methodology, and implemented. Finally, the implementation had to provide tools that help system architects to improve their system design.

The methodology was aimed to work with artifacts closely related to today's architecture and design model artifacts. To be accepted, it should show how performance analysis can be integrated into modern system design processes, allowing it to take a new role in modern software engineering.

In this chapter we will re-visit the aims and scopes of our research, discuss the immediate and derived research goals, the identified and implemented parts, and we will draw conclusions on the degree of coverage our research has reached so far.

7.1 Overall Aims of this Thesis

As mentioned, the aim of our research was to provide a methodology and tools for engineers to learn about a distributed (client / server) system's performance and to be able to build systems with better performance characteristics.

The advent of client / server and distributed systems brought many uncertainties for system architects and developers. Different approaches had been promoted, most of them addressing the ease of programming and structuring. While these issues are of great importance, the introduction of different standards, such as RPC[1], DCE[2], CORBA[3], or RMI[4], only solved the connectivity issues.

As a result there were different approaches following varied objectives: One was minimizing communication since communication was seen as the bottleneck; another one was separating logical modules physically since logical separation and modularization were the only separation criteria well-known until then. Today, other aspects are dominant: E.g minimized deployment effort or the centralization of security critical components.

While all of the issues above are justified and important, little has been done to date to really address the study of distributed systems' performance behaviour at design time. In practice, scenario testing based on worst-case assumptions and benchmarking have been the predominant approaches. It was our aim to provide a methodology with the following characteristics:

- It should be easy to understand and to build models, since the used artifacts are closely related to the artifacts of well-known modeling and design approaches.
- By using discrete event simulation it is based on a well-known and accepted technology to evaluate a system's run-time behaviour.
- It should allow users to compare the performance of different architecture designs plausibly and using their own system usage scenarios.
- It should offer typical procedures for learning about a system's performance behaviour.
- It should be able to apply some performance analysis procedures that experts would perform in a similar way. Doing so, it should also be able to execute some analysis autonomously that would otherwise require the presence of and the control by experts.

[1]Cf. [BN84] for details on the Remote Procedure Call (RPC).

[2]Find more on the Distributed Computing Environment in [Sal96].

[3]Cf. [Gro02] for the Specification of the Common Object Request Broker Architecture (CORBA).

[4]More on Remote Method Invocation (RMI) can be found in [Sun97], deeper information in [MvNV+99].

With tools implementing these characteristics, system architects and designers are in a position to make their design decisions based on systematic analysis and comparison. In addition, they can compare their models using different usage scenarios allowing them not only to compare for an initial system usage assumption, but also to test usage development scenarios and to learn about how a specific architecture fits for change.

7.2 Realizing the Aims

Four our research, as well as the implementation, we chose not to build a new system from scratch but to use existing systems and enhance new functionality reflecting our approach and ideas. A prerequisite was thus to find extensible and controllable tools and approaches covering specific parts of our aims in a way that corresponds with our idea. For example, compared to a black-box tool that provides simulation as closed process from modeling throughout result presentation, a tool following the white-box principle allows us to influence the followed procedures. We want to determine or steer how certain things are handled.

After we started our research it became obvious that we would rely on the tool chain of HIT. The characteristics of HIT have been discussed in-depth in section 6.1. It turned out to be very useful, not only because of its conceptions, but also because some of it is realized as a unix typical tool chain. That allowed us to influence tool activities and insert specific steps between them to realize our own process implementation.

Simulation as Evaluation Technique

We used simulation as evaluation technique since it is able to handle elements of uncertainty, such as probability distribution based implementations, and loops. Our reflections led to the conclusion that an experiment should be executed with a model that is not changed at experiment time. The reasons for this are:

1. It is too difficult to change a model at simulation time with the tools and languages available today.

2. It is difficult to measure the actual effect of a model change at experimentation time, since the many effects of earlier and later phases of experiment execution make causal conclusions virtually impossible.

3. In order to handle the stochastic phenomena correctly and not suffer from their effects, which by their very nature may arrive earlier or later, it is better to handle all experiments in the same way and isolated from each other.

Yet, if a simulation experiment has to be executed individually and isolated from all other experiments, there has to be a way of changing the models and conditions between the experiments in a controlled way that allows to observe the effects of the changes. Our approach departs thus from the idea of executing experiments of models with different changes and comparing the results.

The changes have to be applied in a systematic but efficient way. As individual simulation experiments have to run long enough to cover stochastic phenomena or long-term simulation states comprehensively, there is no potential in shortening simulation time for shorter evaluation. As a consequence, the evaluation strategies have to strive for finding their results in a minimum of experiments.

Efficient Simulation Series and Strategies

As a conclusion of this, a part of our research work has been to build useful and efficient experiment series. The motivation was to find out facts about the observed system efficiently. Our research had thus the aim of showing how to build experiment series that find facts with as little experimentation as possible. This is the point where the techniques of self-adaption have been introduced. These techniques, also known from research on artificial life, have proven to be very useful in our context.

As mentioned above, the basic idea of our approach is learning from model changes. Included in that idea are the definition of a learning target, and the definition of variability, i.e. the specification of how a model can be changed. To identify efficient procedures for the learning targets and the variability as given possibility we have introduced strategies. Strategies aim at reaching learning targets in ways that turned out to be reliable and efficient.

Some of the strategies are made for finding solutions without much knowledge of the problem space. They aim at searching for answers in generally efficient ways. Other strategies are made for specific problems, to find answers and solutions even more efficiently. We will re-visit the strategies in sections 7.3.5 and 7.4.

Tool Chain

Our solution is designed as tool chain or framework. This kind of architecture makes the integration of specific tools easier. The tool chain is aimed to cover the whole evaluation process from model building to the presentation of the results. The tools to be integrated have very different scopes, architectures, and usage interfaces. The implementation of our tool chain had to respect that and realize the integration accordingly.

To show the results of our research we will re-visit the individual parts and shortly discuss their purpose. We will discuss briefly how these parts are realized. And, we will show how the goals of our work have been achieved.

7.3 Specific Aims and Parts of the Evaluation

While the principal aim of our research is a methodology for performance analysis, the visible and tangible result is a set of tools implementing large parts of the methodology using well-known and accepted technologies in combination with strategies for hypothesis testing. Finally, the application of our methodology has to be done in a process that integrates with modern engineering and design processes.

The parts of our work will be re-visited in the next few sections. We will look at their scope and some aspects of the solution they are made for.

The first part is the evaluation process which has to serve as part of an embracing software or solution engineering process. Modeling the system to be observed is the next part; it is based on artifacts that are as nearly as possible related to well known modeling artifacts of the embracing engineering process.

A central part for the evaluation methodology is use of discrete event simulation techniques. An aspect of this is understanding how different

phenomena can be reflected in simulated behaviour, and what kind of control distribution has to be applied for simulating. To go further from individual experiments, the next logical step and thus the next part is the hypothesis driven approach, driving knowledge gaining activities by posing the appropriate questions and having the tool set confirm or contradict the hypothesis.

The core part of our research consists of a set of strategies required for specific problem solving in general performance analysis situations.

7.3.1 The Evaluation Process

The evaluation process is the driver of the performance evaluation. Its purpose is to coordinate the performance analysis activities and to integrate with modern software engineering processes. To achieve this ability, the process must be simple, repeatable, and resumable.

Aim

The aim of the process is to control the performance evaluation.

Necessary parts

The parts necessary for process execution are:

- The strategies to gain information[5]
- The resource and experiment modeling facilities[6]

[5]The strategies to gain information have been covered in chapters 4 and 5.

[6]The facilities of resource modeling and experiment definition have been covered in the chapters 2, 3, and 6.

Characteristics of the process

The process must be executable in pieces, in different phases. The reason for this is the integration of the performance evaluation process with the (software) engineering process. The integration with the engineering process, which makes the evalution process a useful part of it, requires that changes to the engineering model are reflected in changes of the performance evaluation model. To execute the process in pieces or steps demands that early findings are available as base information for later execution steps.

Our evaluation process is driven by hypotheses and corresponding strategies that offer ways to test such hypotheses. The selection of hypotheses to be tested and strategies to be followed is the task of a so-called meta-strategy, which has not been implemented explicitely.

7.3.2 Modeling

Modeling is required to make ideas and conceptions formal and to allow for tools to process them. As a foundation, a theoretical model of well-defined objects, relationships, and their specific characteristics has to be introduced and used consistently.

Aims

The aims of modeling in our approach are:

- to concisely and understandably formalize the hierarchies of resources from a view of reality or from an idea
- to connect the resources with each other and to implement their services and usage patterns
- to specify the variable parts and to implement the constraints controlling the variability
- to define the experiment and its control parameters
- to define observation points and select the criteria for their observation and reporting, and
- to allow for users to specify their interests and hot spots, their directives for the evaluation.

Parts

The parts required for these aims are:

- a graphical editor for resource modeling including the resource processing properties (speed, concurrency), connections between resources, and service implementation[7]
- an experiment parameter specification facility
- a control point definition and specification facility
- a facility to define variability and constraints

Another facility to define hypotheses and chose strategies is not seen as part of modeling. Although the evaluation model may contain, some time later, indications on what the evaluation system should seek for, directing the evaluation is today seen as control activity.

7.3.3 Simulation Techniques

There are many forms of performance behaviour observations that can serve as base information for the performance analysis of an entire system. Some forms, such as defined and determnistic algorithms, allow analytical studies of the performance behaviour. Others forms such as observation of average behaviour, programming patterns like conditional execution and loops, or probabilistic statements make it impossible to apply such analyses for all systems.

As an alternative, simulation techniques make system evaluation possible even for very complex model structures. Discrete event simulation is a feasible approach to evaluate a system efficiently, since all forms of probabilistic and conditional statements can be expressed and the observation is reduced to the state changes. Even rare, improbable events happening once in a while can be considered appropriately – and their effect respected adequately – by defining a duration of simulation time that respects these circumstances.

[7]The term *implementation* does not refer to a real implementation in this case, but rather means performance behaviour in terms of time consumption and consumption of other resources' services

Aim

The aim of using discrete event simulation techniques is thus both to be able to analyze models with all possible kinds of behavioral specifications and to run the analyses as efficiently as possible.

Parts

The tool used to run the simulations was HIT™, an evaluation tool of the University of Dortmund. It offers a graphical editor for model building and specification and an evaluation language used to translate the models and experiments into Simula. The graphical editor Hitgraphic™ is provided by HIT for the specification of resource models. An individual experiment can be defined an ran from the Hitgraphic application window. We use Hitgraphic's ability to generate HI-SLANG experiment definitions including the resource model definition and the experiment parameters.

As individual experiments are parts of strategy executions, they contribute to the strategies' information gaining process. As such, they are aimed at providing statistical information for a specific configuration of a strategy execution. In a fully integrated monolithical application the experiment execution would yield these data as part of the information of a bigger structure. However, as HIT has no notion of our strategies, the information is reported by HIT and collected later by a pattern matching script that provides the information for the strategie's data collection.

7.3.4 Hypotheses

Aims

The aim of hypotheses is to drive the performance evaluation into a specific direction. The two simple patterns are:

1. We assume that varying a specific parameter x has influence on the performance achievable with the given base model. How much influence does a specific change have? If we apply the change multiple times, is the effect on the performance behaviour always the same or does it become more or less?

2. Given such a model variable x, is there a marginal value for x where any further variation does not have any more positive influence on the performance behaviour?

Parts

Today, the hypotheses are not formalized as system parts. Their usefulness comes from the theoretic consideration on what questions should be looked at and how they could be solved. Each hypothesis has its direct reflection in a strategy though.

7.3.5 Strategies

Aims

To address the questions raised by the hypotheses in a structured way, strategies have been introduced. A strategy is aimed to implement a problem solution in a general, efficient way.

Parts

In general, the strategies consist of specifications of variable parts of the model, including the notion of constraints, a specific hypothesis and an algorithm addressing the hypothesis, and a termination criterion indicating whether the hypothesis has been examined sufficiently or not.

Every strategy implementation is implemented as part of the evaluation workflow framework. It must be implemented in a way that the evaluation can be started, re-started, or resumed at any particular point. Notice that this does not hold true for the used instruments. An interrupted simulation experiment or modeling activity has to be executed again for most cases.

A part of strategy implementations is the preparation and execution of individual experiments.

7.4 Performance Evaluation Strategies

The strategies are an essential core of our work. Their aim is to define procedures for an evaluation tool to find information about the run-time behaviour of a system model if specific requirements are met.

During our work we have investigated strategies with different scopes. Each strategy is aimed at finding out information by evaluating in a specific direction. A hypothesis is the driver for each individual strategy. It is addressed using practical experiments and other sources of information. The strategies' purpose is to confirm or contradict the hypothesis, or to locate a dividing line between a specific yes and no.

The answer to a hypothesis is not a proof. Due to the stochastic nature of both models and experiments, it is possible that a specific phenomenon appears in one experiment execution and does not appear in one with exactly the same parameters. Our strategies do thus have to ensure the reliability of the identified information with high probability. An experiment is thus a test to be executed in a way that the stochastic phenomena are covered plausibly.

7.4.1 Hypotheses — Strategy Questions

The hypotheses used to learn about a modeled system's performance follow very few base patterns. Here are some questions covering these patterns:

- *Given a system model and a workload type applied to that model, how much workload can the system process as a maximum?* In other words: *What is the maximum average workload insertion rate for which the system works in a stable way?*
- *Given two models for a target system, which model exhibits better performance?*
- *Given a model of a system with two different, independent types of workload, and given that the two workload types are not comparable and adding their number does not give a meaningful result for a performance indicator, how can maximum performance be characterized? Where is the performance maximum? Where are the maxima of each individual workload type? How sensitive is workload type A to changes in the*

insertion rate for workload type B? Is the sensitivity significantly different for different relationships between the insertion rates of A and B?

- *Given a model with the performance maximum known, how much does the performance scale if more resources are added to the resource model, i.e. how much can the performance maximum be augmented? How many times does adding resources augment the performance? What is the 'absolute' performance maximum?*

- *Given two models for a target system, both with similar performance characteristics, which model scales better? Which model arrives at a higher performance maximum for maximum scaling?*

7.4.2 Some Strategies Revisited

The Cold Start Protocol

The Cold Start Protocol was introduced in section 5.2.1. It aims at locating a maximum value for workload insertion still leading to valid experiment execution. It is based on the assumption that even with stochastic effects the performance maximum can be localized within an interval of workload insertion rate values. Its mission is to search the interval efficiently and to narrow it as much as desirable to get a relatively reliable value assumed to be the performance maximum.

Multi-Dimensional Maximum Localization

The modified Cold Start Protocol for multi-dimensional localization of the performance maximum was introduced in section 5.3.2. It uses the principles of the Cold Start Protocol to localize a maximum value in one direction while keeping the mixture relationship between the different types of workload fixed. A so-called Warm Start uses the approach of the Cold Start to localize the maximum value searched for, but starts its search from an identified maximum value nearby.

Scalability for Models with one Workload Type

We have introduced our concepts for scalability analysis in section 5.5. The easiest case of scalability analysis examines models with one type of workload: There are two variable items for such a model, the workload insertion rate allowing us to identify a performance maximum for a given configuration, and the variable resource pool or combination of resource pools. Varying the latter allows us to learn more about performance gains possible with resource enhancements.

7.5 Research Results

In this section we will re-visit quickly the different achievements of our research and illustrate how far our research results go.

7.5.1 Theory and Foundation

Our research is positioned in the field of software development and development processes. It is aimed to address specific questions of performance and to give designers the tools to find answers to these questions. A specific understanding of objects, systems, conceptions, and behaviour is required for the people who work with these tools.

The foundation of our methodology is a theoretical model of objects, their usage, their mutual relationship, and time. As a base, we have introduced the elements resource, resource type, resource pool, workload, location, co-location, resource consumption, and a few more. Then, we have discussed the observables known a priori and a posteriori, i.e. before and after the evaluation. And, we have shown what detailed information can be drawn from what kinds of individual experiments or experiment series.

Another section was devoted to the questions to be addressed in our context. We have translated the questions into problem statements composed of the terms of our theoretical models. After that we were able to show ways of finding answers to some of the questions by executing specific experiments or experiment series. We found the variation of specific model parameters to be the key to most of the solutions and at the same time an indicator of problem complexity. Every variable parameter means one more degree

of freedom, and every degree of freedom means at first one more order of complexity and of evaluation time.

In the following work we have permanently sought to find adequate techniques to reduce complexity and to make evaluations faster but without losing significance or preciseness. One way to do so and to even gain expressiveness is the definition of constraints. Constraints are used to define valid combinations of specific values, i.e. to restrict the variation in a way that parameter combinations are only admitted for experiments if they are plausible for the model.

7.5.2 Process

The way of addressing and solving problems in performance evaluation led to the definition of a performance evalution process. The process defines a general procedure of executing an evaluation and using the available evaluation instruments but also of collaborating with other engineering processes. The aim was to make the evaluation process not only compatible but also integratable into software and system engineering and design processes. In our work, we show the structure and composition of a performance evaluation process that follows our recommendations. The recommendations are the following:

1. Simplicity and comprehensibility
2. Possibility to execute the process with interruptions
3. Definition and manipulation of artifacts similar to the ones used for design modeling
4. Possibility to build scenarios and to run hypothesis-driven evaluations
5. Coverage of standard procedures for standard hypotheses

The implementation of our process is aimed at integrating specialized instruments. Each of them addresses a part of the evaluation which is necessary for the whole process, but which is almost independent from all other parts. In the current implementation, the instruments cover a specialized task in an isolated way. They do not fully integrate with each other, neither do they all work with the terms defined for the performance

evaluation. Because of this, there has always to be a user who executes the process.

However, we do not consider the current state of implementation as a disadvantage. The instruments have been chosen and used due to their unique contribution to the overall process. Some instruments have their origins in other usage or tool combinations and have to be used appropriately. For example, the modeling instrument is made of Hitgraphic, the graphical modeling and experimentation tool of HIT; design models built in other tools have to be translated manually into a Hitgraphic model, and changes have to be brought in manually too.

Other instruments have been implemented explicitely for our scope and fit better into the current process implementation. For example, the mechanisms of workload variation are part of our strategy implementations; after a few checks by the users they operate autonomously and without user support. The target, however, should be to have a fully integrated, potentially fully autonomous process implementation some time later.

7.5.3 Process Instruments

As mentioned in section 7.5.2 the process is only implemented in parts, namely as process instruments[8]. The following areas have been covered essentially:

1. The modeling instruments
2. the evaluation instruments
3. the instruments of self-adaption, and
4. instruments for user interaction

Some of the instruments, such as the evaluation instruments or the self-adaption instruments, are part of our core research. They will be illustrated in the next few paragraphs again. Others, such as the modeling instruments e.g., have been used largely the way they had been available from the integrated tools.

[8]The notion of process instruments has been introduced in section 3.2.

Not much emphasis had been given to modeling according to our approach foundation. Since we had decided to use HIT in a very early phase, it had been clear for us that we focused on the conceptions that were expressible with HI-SLANG and that we would not spend much energy in enhancing the capabilities of Hitgraphic in graphic modeling. Similarly, we have had the experiment definitions represented in HIT, leaving only the management of parameters and information on the higher levels, i.e. on the strategies level, for our complementary software. Yet, we had still a very powerful graphical modeling tool with Hitgraphic and another powerful language based tool with HI-SLANG and HIT's HI-SLANG compiler. Using these tools, we were able to encode large parts of our evaluation models.

The models we wish to analyze have specific characteristics, such as conditional execution statements and loops, or dependency on probability distributions. This makes simulation virtually the only examination technique for an individual experiment. The instrument for simulation execution consists of a simulation engine delivered by HIT and an encapsulation that makes it usable for our implementation and allows our specific examination aspects to be included appropriately. The encapsulation is responsible for the preparation of a HI-SLANG experiment fitting into the experiment series of the corresponding strategy and for the collection of the HIT generated result values. It offers a kind of abstract simulation engine interface to the logically higher levels, and it controls HIT's simulation engine below.

The instruments of self-adaption are the means of learning for the evaluation process. They are used for different aims, but they always help towards aggregating information about a system's behaviour under load. Self-adaption techniques are in most cases used for step selection as part of an efficient strategy execution, i.e. specification of the next step, typically the next experiment, based on the current state of information and of progress in strategy execution. If a target system has to be modified, e.g. as a consequence of a model change in the design process, the changes have to be reflected by the users. An improvement of the evaluation process here would use mechanisms of self-adaption to be able to draw analogies between the model before and after the change. Using such mechanisms can help avoiding to re-execute all experiments executed in the previous model state.

Aggregation is also a technique for improving the evaluation efficiency. Its aim is to replace specific structures of an original model in a way that the original structure's run-time behaviour is preserved while the simulation is simpler. The new object is capable of processing all workloads that were previously inserted into the old structure, and of consuming the underlying

resources in the same way. Only workload generation and processing within the previous structure is eliminated and this simulation time is thus economized. Supported by corresponding abilities of HIT, we were able to realize this functionality of our tool that may make evaluations much faster.

The instruments of user interaction are the ones that have been developed the least. Most important for our research was to have presentation graphics for our strategy results. We have essentially identified ways of presenting identified results of an individual strategy. It is important for graphic representations to present the results clearly, allow for easy interpretation by the users, but not to suggest misleading conclusions or erroneous interpretations.

The graphic representation of the evaluation activities at run-time, or the possibility for users to influence the evaluation based on such a graphic representation have not been studied deeply. At the moment, the strategy implementations are chosen manually and executed in an isolated, autonomous way. We see much potential in changing the evaluation to be a more interactive task with the possibilitites to be observed and influenced at run-time.

7.5.4 Variation and Strategies

Most of the information gained about the performance of a modeled system has been found with techniques, of which the base principle is the variation of some specific element of the experiment model. Much can be learned about the role of an object in the context of the entire resource model if one of the object's parameters is varied. The varying results in experiments show the influence of that parameter for the system's performance.

The first and most used form of variation is workload variation. Since the workload is inserted by the insertion object in specific patterns, the variation is applied to a parameter of that pattern. In most cases, this has been the so called workload insertion rate[9]. We have presented models with one type of workload and others with multiple independent types. For workload variations, the number of these independent types defines the dimensions of the problem or domain space. Whether there is only one type of workload or whether there are multiple types, the workload for the system is changed by

[9]In our simulation environment the insertion rate is manipulated by changing the interarrival time, i.e. the mean time between two workload arrivals.

varying the workload insertion rate. Our procedures have thus to be made in a way that they can navigate through the domain space and find solutions efficiently.

We have shown that this kind of variation brings the evaluation system in a position to be able to approximate a performance maximum. We have also shown that the performance maximum is not a value that can be determined clearly and sharply but rather a zone, and that the maximum for multiple types of workload is not a simple value but rather a region in the corresponding solution space. Such a zone or region can be represented and shown using a specific kind of graph in the solution space. For the case with one workload type we have investigated the approximation possibilities and we have presented a general way of localizing the performance limit. In addition, we have introduced an efficient method of localization called the Cold Start Protocol. And, we have shown the consequence of working with an approximation zone by showing techniques of not only approaching but also narrowing the zone.

A way of evaluating the maximum performance has been illustrated for the case of multiple types of workload. For the two-dimensional case a general solution method has also been introduced and implemented. We have shown that the Cold Start Protocol was a good principle procedure to depart from, but also that is was not efficient to execute the Cold Start n times. The Warm Start protocol was introduced as an alternative for the Cold Start, enabling the evalution strategy to re-use the previously identified result as starting point and gaining evaluation efficiency this way. This protocol turned out to be very useful for multi-dimensional variation, since the results for a nearby problem space are often close to the nearby results[10]. Finally, we have also shown with which principles the navigation in the problem space can be made in an efficient way.

As a next result, we have presented the comparison of models based on performance maximum localization. Using this systematic comparison approach, system designers are able to get an impression of the consequences of a specific design choice.

A special and systematic form of model comparison is the model variation for scalability analysis. We have addressed the questions in this context to show suitable procedures for scalability studies using our approach. Our research showed that although our approach is very time consuming our tools

[10]This could be looked at as a locality principle; yet there is no guarantee due to the unknown form of the solution region and to the stochastic nature of the resource model.

allow users to investigate scalability in a way that has not yet been followed much. We are convinced that this form of performance considerations will gain much more importance for system designers in the future.

To implement our strategies we have realized a framework inspired by the principles of workflow technology[11]. A premise of our implementation was the possibility to define and execute a complex sequence of activities, not necessarily fully defined a priori, in a way that the sequence can be interrupted, aborted, stored, and resumed at virtually any time of execution. Our implementation in Perl unites these characteristics with the reliable operation and integration of other tools and instruments. Our framework turned out to be very useful this way.

7.5.5 Evaluation System Architecture

To build an evaluation system that implements our proposed evaluation approaches, an appropriate system architecture is required to reflect the individual procedures of evaluation and to allow and support the integration of specialized tools and components. The facilitation of the flow of information and of the control realizing the collaboration of the specialized tools are essential for a successful implementation.

We have proposed a general architecture for our system in section 6.4. We have shown that it is important to integrate the individual tools on the right level of abstraction or level of purpose. In addition, there are tasks covering more than one area; we have shown this in figure 6.10 for the Modeling Control. It must be possible to integrate even these tools into a suitable architecture. Likewise, if a tool would ever be sold commercially, an integrated control and graphical user interface component would probably cover the whole evaluation process; yet the integration and information flow requirements remain the same.

We have been using a meta data structure facilitating the collaboration of components. In that data structure we have kept all information which was not part of the data of the specialized tools; the flow of information between the different activities of the value chain has been realized in this structure. However an integrated information flow facility transporting an information network has not yet been implemented.

[11] Cf. [Hol95] for the specification of the workflow standard, or [vdAvH02] for more information about workflow technology.

7.5.6 Presentation of Results

Until today, the results that have been found by any step of our evaluation tool are collected at the end of that particular step; the step then generates a representation suitable for interpretation and understanding by users. The graphic presentation is the typical representation. A good picture presents the results or the interpretation in a way that it is easy to see and to understand by users. To do so, we have especially respected the following criteria:

1. Statistic values rising or falling strongly with the variation may be represented better in a logarithmically scaled graph. In that graph it is easy to see discontinuities or deviations from the expectation.

2. Values of a row with only minor differences should often be represented as equal, especially if the reason for inequality may be found in the stochastic nature of the respective experiment.

3. It may be crucial to represent the results in the light of important context to see whether critical events have taken place or not. E.g. a graph of the throughput and average of pending requests for a specific component may help users detect that this particular component plays a bottleneck role.

The presentation of the results is in our approach tightly coupled with the individual strategies. This comes from the fact that the context and aim of the experiment series is obvious for the individual strategy execution, but is less obvious outside. For that reason, we have implemented the activities to collect the HIT generated result files, to extract their result values, to save them, and to build a graphic representation as integrated part of each strategy. Collecting and extracting such information is required by most of the strategies anyway, since their step selection mechanism relies on the data identified earlier. Using that data for the generation of a typical graph is thus only a little step further.

In many cases we have selected a way of representing results graphically that is prototypical for the results of the executed strategy. As the aim of such a graph is to make interpretation and conclusions simple and compelling, the critical facts have to be made visible in a typical way so that the users get used to the image. Since at this time of development the experiment strategies are

the only mechanisms to gain information about the behaviour of a system, it was enough to embed that capability into the strategy implementations. If, however, there will be novel approaches to combine the results and to gain information from them, especially if that happens without explicit execution of experiments, new ways of representation may be required.

7.6 Potential for Further Research

In this section we will introduce some points of our approach of which we think they have great potential of improving the approach considerably. We see immediate improvement potential in the following areas:

- Process implementation and meta strategy
- Interpretation of results
- Proximity of experiments
- Information exchange within the process
- Improved user interfaces

7.6.1 Process

The current implementation gives users only the evaluation instruments that allow them to execute specific parts of the process autonomously. The full process is still human-driven. The user has to choose a strategy, prepare the evaluation model accordingly, run the execution, and run the results preparation separately.

We do not consider this state of development as a disadvantage, since there are steps that have to be, by their very nature, executed manually. Amongst others there is e.g. the step of transforming a design model[12] into a evaluation model according to our approach[13], and every change to the design model has to be transferred to the evaluation model as well. But there are

[12]The design model is the product of the design process for a system.
[13]Model building according to our approach was introduced in chapter 2.

also parts of the process that could be automated fully in a way that the process could perform its default evaluation behaviour autonomously.

We see great potential for our approach in a so-called *meta strategy* that automates a default procedure with the most important elements of process control. The meta strategy's aim is to select and execute individual strategies according to users' directives[14] and based on the currently available information.

The advantage of such a meta strategy is the ability of an evaluation tool to pursue larger parts of the evaluation with more autonomy. After an initial modeling phase and the specification of directives indicating what evaluation goals should be examined the meta strategy chooses individual strategies autonomously, executes them, collects and interpretes the results, and prepares for another strategy selection. The consequences of such an enhancement could be:

- More autonomy for the tools
- The tool is able to run the less important but computing-intensive tasks without permanent control of an expert
- For laymen, the convenience of operating the tool is increased largely, since less steering by them is required and more is delivered by the tool

We think that implementing meta strategies should be made in different ways. One approach is autonomy-oriented. Its aim is to autonomously and efficiently select and execute strategies in a way that crucial information is identified as soon as possible. The other approach is interaction-oriented. The aim of this approach is to support the user in strategy selection as effectively as possible.

These meta strategy approaches are not necessarily completely different. While the former one should be optimized for highly effective strategy selection leading to an efficient evaluation, the latter has to make recommendations for good strategy selections. In fact, if the user consistently follows the path of most recommended strategies, a very similar evaluation path is quite plausible. In addition, there could be mixed implementations, and switching amongst the chosen meta strategies could also be realized. This all allows

[14]Users' directives include indications about what the evaluation should test and find out and what direction should rather not be followed.

for a performance evaluation system to support its users in the best possible way.

There are approaches for such a meta strategy with origins in different scientific areas such as self-organization and self-adaption theory but also dynamic workflow theory. Dynamic workflows are composed at run-time meaning that their steps are not known or pre-defined when the workflow is started; they are determined in activities that are part of the workflow[15]. Prerequisite for a successful implementation is the capability of well-working information transfer.

7.6.2 Interpretation of Results

Our current implementation of strategies is realized in a way that each strategy produces the required experiment results itself. In theory, it is possible to produce individual experiment results for one strategy and to re-use the results for another one. Such an improved implementation would make the evalutation considerably more efficient, since double experimentation could be avoided. To make that possible, the following criteria have to be met:

1. The preconditions and the context of the experiments must be comparable.
2. The semantics of the results and measurements must be defined concisely[16].
3. The strategies must be implemented in a way that they can assess analogies and make use of earlier experiments on their own.

For an automatic interpretation of information, we have to adress both the appropriate means for information exchange and the extension of the strategy implementation. The extended strategy implementations must be able to examine existing results under the perspective of its own scope with

[15] As a matter of fact, we have used this technique for our implementation already.

[16] Interpretation procedures that are part of a specific strategy are realized in and for the context of their strategy. Interpretation procedures, which re-use earlier results, are not realized in the light of a specific strategy. Because of that, they have to use much more concise context information for the results to be interpreted. For example, a series of results for an experiment, where only one parameter changed, is not necessarily an approximation series; analyzing the series in that way would lead to a fatal misinterpretation

the aim to optimize its own performance. The optimization in this case means that the mechanisms of self-adaption are used and improved in a way that equal evaluation results can be established in less time.

The interpretation happens in different places of the evaluation. First of all, every strategy uses interpreted results in one or another way. Some strategies require the results of earlier experiments to determine their next step; that could also hold true for results of experiments not executed by themselves. In fact, the strategies may even analyze whether experiment execution is necessary for a step or not. Then, the strategy also produces results by interpreting the available results of the experiment series; there, the use of results fitting into the row of experiments executed by the strategy should also be included.

We have shown earlier that the interpretation of one strategy is a result that may be used by other strategies for further interpretations[17]. As (higher-level) results, the interpretations have thus to meet the criteria listed above for results in general. They must be comparable with respect to preconditions and context, they must be defined concisely, and strategies must be able to use them for their own purpose[18].

7.6.3 Proximity of Experiments

Another approach seems to be promising for accelerating large evaluations: Instead of running an experiment that has not been executed before, the results of a previously executed experiment may be helpful too. Such a technique would rely on the fact that an experiment does not exactly meet the specifications of an experiment chosen for execution by a strategy, but is close enough that the chosen experiment needs not to be executed.

The critical term is proximity, and the criterion is whether the specification of an executed exeperiment is close enough to the currently required experiment specification that analogies are possible and useful, and that the execution of the new experiment can be avoided. However, it is very difficult to formalize such a term of proximity due to the following reasons:

[17] E.g. a performance maximum determined with the Cold Start Protocol is taken as starting point for more performance maximum evaluations for similar resource models in the scalability analysis.

[18] This is only a normative statement. Of course not every strategy is required to be able to use every kind of results.

1. What can be ignored and what cannot be ignored? If the results of two experiments are very much different there is obviously enough reason to have both executed. The same holds true for totally different models. On the other hand, if two models are almost the same, we do not know whether or not the difference is critical. If the change concerns a model component that has been critical for the performance of the model, e.g. as a bottleneck, it is probably advisable to execute the experiment with the changed model. If the changed component has not been critical until then, we can still not assume that is will not be critical in the future.

   Consider the example of a model taken from a distributed system design process and examined somewhat. Changes of the design process model have to be reflected in the performance evaluation model. Is the experimentation executed until now worthless? Do the strategies have to be executed again or is there a way of finding analogies and shorten the new evaluation by using the results of earlier evaluation steps?

2. How close is 'close'? In the Cold Start Protocol introduced in section 5.2.1 there is an approximation phase and a narrowing phase. For the approximation phase it is important to roughly localize the validity limit; in a series of exponentially growing or shrinking workload insertion rates any experiment with another insertion rate but with all other parameters unchanged is considered helpful. The proximity term would be interpreted widely. In the narrowing phase, however, the proximity of the series becomes greater and greater; a too widely interpreted proximity criterion might even hinder the protocol, since, due to earlier steps, an identified step might be 'avoided'.

   We learn that proximity is a relative term even with respect to one parameter and within one sole strategy with all other parameters left unchanged.

3. How much result deviation justifies another experiment? As indicated above obviously differing results are in most cases a reason to execute a new experiment. On the other hand, similarity of results is not a reason for not executing an experiment. Complex performance evaluations, e.g. for the examination of scalability analysis, rely on the comparison of very fine differences.

We think that the criteria for proximity have to consider both the expectation for and the critical role of experiment selection and execution.

One possible approach is to determine the expectation of the effect of changes to a model as a first indicator. If there are major changes expected in the examined context, proximity can supposedly not be assumed. If expectation analysis indicates that there are only minor changes, we have to determine whether the experiment plays a critical role in the strategy's experiment series as a second indicator. If the change is critical in the context of the evaluation stage, e.g. because the current phase of the strategy demands it, proximity is not assumed either.

We are convinced that serious work on this issue will lead to techniques that help to accelerate performance evaluations significantly. This holds especially true if they are executed as integrated parts of an embracing engineering process.

7.6.4 Information Exchange

A key to the success of a complete and efficient process is the usage of every kind of information for interpretation, conclusions, and strategy selection. During our research we have seen the following classes of information:

- The resource model including the resource types, instances, pools, and services as well as their connections for respective service usage, i.e. the generation of workload of one for another one.
- The experiment model including the resource model and the experiment parameters required for the respective tools, e.g. the duration of an individual experiment.
- The evaluation model containing definitions of variability and constraints as well as directives or indications of what information is required.
- The evaluation results of previous experiments and strategy executions; results are both measurement values and interpretations.

The exchange of evaluation results has promising potential for the improvement of our evaluation approach. While the current implementation exchanges information amongst experiments within an evaluation strategy thread, the sharing between different strategies has not been addressed

effectively. Most strategies are executed with a minimum of information available at starting time.

As mentioned in section 7.6.2 the exchange of information must happen in a way that the quality of information can be assessed by every component that makes interpretations or conclusions. Comparability of the quality is a core issue; an individual statistic variable, say e.g. the throughput of a resource pool, may be worthless if the context or even the scope or aim of determination is not clear. The context information for a specific result value is thus a prerequisite for information transfer.

In section 7.6.2 we have listed the conditions for the usefulness of automatic interpretation of results. For a strategy saving the results of an experiment, it does not only save the found experiment values including the precise definitions of the measurements, but also the whole experiment model is part of the information to be saved. If the evaluation engine did save the whole information for each experiment, the stored data would grow to huge amounts. The storage should rather be organized as a whole, allowing each experiment to store only the particularities compared to the initial base model. The amount of information from which we will draw conclusions is relatively small compared to the amount of context information; but without the context information it is more or less worthless. We call the first kind of information the *annotated information*, while the context information is called the *annotation to the information*.

The information exchange realized today is building series and "overall results" by individual strategies. The execution of the Cold Start Protocol, e.g., results in an approximated value indicating the modeled system's performance, which can be the overall result sought for. Our understanding is that this result is a higher order result that was found by executing an entire series of experiments. Our implementation has already a minimum of capabilities to build results from series of experiments. Theoretically, it is not necessary to always execute a specialized strategy to get information out of individual experiment results; we can also think of analysis components able to create new interpretations of available result values and thus able to generate new results. We think of an approach that uses techniques of the 'data warehouse' / 'data mining' area[19], well-knowing that we require to get more information out of few data while the latter is aimed at getting more information out of a huge amount of data. Especially with

[19]Learn more on techniques of data mining in [Kan02], [Dun02], or [HK00].

novel interpretation techniques, the information is required to be examined together with its context.

Our idea is to set up a network of annotated information during the course of an evaluation. The network structure comes from the fact that higher order results depend on the results of lower order; we can regard them as relative results. If the continued evaluation produces results that would lead to different interpretations[20], the higher order results will have to be revisited. A reason for this is the fact that the results of all orders are often only approximated under the premises of a defined preciseness.

If the information is structured and passed further in the form of such a network of annotated information, the evaluation becomes a kind of information bus. Each component corresponds to a filter that examines and adds data, regardless whether this happens with or without executing experiments. Other than for other cases of information bus conceptions[21] or approaches of data warehouses and data mining, the base information out of which the higher order information is derived is only created at execution time. New base knowledge can come out at any time of evaluation. If a mechanism can be found that handles these challenges successfully the evaluation can be improved considerably.

7.6.5 Human-Computer Interface

In our research we have mainly focused on showing ways and approaches to determine results of performance evaluations correctly and efficiently and to represent the results in a way that users can make the right interpretations and conclusions. To allow users to bring in the resource model and parts of the experiment definitions graphically, we have integrated Hitgraphic, a HIT specific modeling and code generation tool, into our tool chain. All other action had to be done without graphical user interface, partly as command line operations, partly even as modifications to existing scripts. As a consequence, the resulting implementation of our research is not an integrated, mature, GUI enabled application but a set of different tools and scripts. It is probably too complicated for laymen to be used as an every day tool.

[20] A further experiment result might produce a closer approximation, or a confirmation or rejection for a tested hypothesis.

[21] Cf. [HMW02] for more on such information bus conceptions in the Web Mining context.

We are convinced that a good user interface will drastically improve the acceptance of system performance evaluation tools in the future. To do that, a tool must adhere to the following prerequisites:

- There must be (graphical) user interfaces differenciated for experts and for laymen.
- It has to make the evaluation results available during every phase of evaluation quickly and in a comprehensive way.
- It has to guide the evaluators through the evaluation process by visualizing the followed paths of evaluation and experimentation and by proposing the next steps similarly.
- It has to somehow support the analogies between performance modeling and design modeling.
- It has to be able to show the correlation between the identified results and the applied strategies.

Evaluation Paths Interface

Evaluating the performance of a system can be looked at as exploring an unknown landscape. The variables of our models can then be interpreted as the navigation directions and the problem space as the unexplored country. If there is a way of expressing the problem space as a theoretical landscape, there is a big potential for using techniques of geographic information systems or systems management software[22].

If a graphical user interface can be built that represents the navigation in that space by the strategies as paths through the problem space, it will probably be very intuitive to use. Open paths that could be chosen by users could be symbolized in the landscape by lines of a specific color.

If a good meta-strategy is followed, the possible paths could also be plotted in different colors according to the probability of selection for the next strategy step. This would allow users to identify what the evaluation system will do next and experts to react to strategy selection before it has actually happened.

[22]Some analogy can be found in approaches for the visualization of software evolution(e.g. in [CKN+03]).

The manipulation of these paths may allow users to re-value the probability of path selection by attributing a correction factor. Notice that different paths represent different strategy selections but not necessarily different strategies. A strategy selection represents a strategy for a very specific context while strategy is only the word for a generally available algorithm or pattern.

Critical Component Interface

As soon as some evaluation has taken place, there is in most cases information about key values at specific components in the model. A graphical user interface may represent the resource model with the resource pools in colors depending on their role in the overall performance results. For example, if one resource pool was the limiting factor for a performance maximum evaluation, that pool could be marked red and others green or blue to show to the users which is the critical component and what should be looked at. Coloring is best done gradually. With gradual coloring it is easier for users to recognize whether the critical component is the only one having a limiting role or whether other components may also impose limitations.

Aggregation Interface

A graphical interface could also be useful for aggregation[23]. It should be possible to mark the components to be aggregated and to have the evaluation system build the aggregate component. The studies and analysis to build the aggregate component should be made by the system automatically. The system should also maintain the correlation between the original model and the model with the aggregate component.

[23] Aggregation was introduced in section 3.2.3.

Appendices

Terms of Performance Evaluation

Assessment Function: An assessment function is a function aimed at making an assessment of identified results possible and, in many cases, comparable. It is required for many performance evaluation strategies, especially localization strategies.

Capacity: Ability of a resource or resource pool to process a specific amount of workload elements within a specific period of simulated time. Capacity enhancements are either realized by making resources faster or by adding more resources to a resource pool.

Notice that it is often necessary to change the workload implementation for enhanced capacities, since many applications in the real world require more coordination effort for enhanced capacity. Such changes are typically implemented as constraints.

Configuration: Specific extension of a model with variable units. For a model with different degrees of freedom a configuration means that one specific value for each dimension is chosen and that an experiment is executed with these values given.

Constraint: Conception required to constrain the variation in different dimensions to specific combinations of values. Conceptions may be realized as formulæ, positive value sets, or sets of undesired values.

Constraints are typically used to reduce the exploration to combinations of values that are plausible for real-world situations.

Discrete Event Simulation: Simulation of a model's behaviour for a specific simulated time. Discrete event simulation assumes that the model has a controlled, determined state at every moment of

simulation, and that state changes can only happen at specific moment by issued stimuli called events.

Many events in a period of simulated time require typically a longer duration of the simulation. Discrete event simulations are not deterministic since they can rely on stochastic elements of modeling.

Instrument: A part of process implementation with specific objectives. An instrument is specialized and is often only useful if applied in combination with other instruments. Its aim is to implement a specific explicit conception of the evaluation process.

Key Criterion: Criterion for a decision (function) to lead to a yes or no / true or false decision. In this work, key criteria are based on key value to fixed value comparison. Typical criteria are: a key value is above or below a specific value, a key value is between two boundary values, the relationship between two key values is greater than a specific value, and so on.

Key Indicator: Is used as a synonym for observed value. The term 'key performance indicator' (KPI) is often used in service management approaches[24].

Key Value: Is used as a synonym for observed value.

Location: Specification of the resource's (geographic) position with respect to other resources. Only resources that are co-located, can work with each other, i.e. the higher level virtual resource can use the lower level resource, if they share the same location. Location can also be an extent of geographic locations, to represent connecting elements, such as a communication network.

Localization: Activity to find a margin value. Some hypotheses do not require confirmation or contradiction. There is rather a determining dimension for a desired result on which the hypothesis testing strategy has to execute variation in a way that the range with the desired result is determined.

Localization strategies are often used to find a maximum or minimum value, e.g. the performance maximum using the validity function as criterion.

[24]For more on service management, please refer to [oGC00] or [oGC01].

Observation Period: Period of the experiment, during the system or system part is observed and statistic information for observation values is collected. Often, the observation period corresponds to the whole experiment duration. However, if users are interested in the behaviour during a specific period, typically in the midth of the experiment, sometimes at the beginning or at the end, defining the observation period is a handy approach.

Notice that the observation period should be long enough to allow for evaluations to get enough data to determine observation values with statistic relevance.

Observation Point: Resource pool or whole resource model, for which observation values are measured. An observation for the whole system theoretically refers to all workload elements inserted into the model. A resource pool observation point is only suitable for the workload elements inserted into that particular resource pool, i.e. only for calls to its resources' services.

Observed Value: Statistic value or measurement expressing a specific quality. Observed values are often results of experiments or experiment series or a statistic evaluation of such results. They are aimed as indicators for the average state of a model.

Typical observed values for experiments are throughput, turnaround time, occupation ratio, number of workload elements in the system, or the number of workload elements introduced and not finished at a specific resource pool.

Performance: Generic term of the processing speed of a system. Performance itself is not measurable; performance measurement thus refers to measurement of key values, such as throughput or turnaround time.

Scalability: Ability of a system (or model) to increase its performance with added resource capacity or to lower it with reduced resource capacity. As different resource pools contribute differently to the performance behaviour of a system, not every capacity change to any resource pool has the same effect.

Simulated Time: Also referred to as *simulated time period*. The period of simulated time – opposed to real-world time – an individual experiment reaches or should reach within the constraints of simulation execution. In many cases, a target value for simulated time indicates a minimum of simulated-world time that a simulation should cover in order to deliver

statistically relevant data. Given a fixed amount of simulation time (real-world time), the more events happen per unit of simulated time, the less simulated time is spent or reached. In other words: The less activity in a model, the more simulated-world time can be spent during that experiment.

Simulation Time: Also referred to as *simulation period*. The real-world time duration of one simulation experiment, i.e. the time it takes to run 1 simulation experiment, excluded code generation and compile time. The simulation time is often specified to prevent an experiment to run arbitrarily long.

Strategy: Conception and implementation of a procedure able to run a hypothesis test. Strategies implement specific tasks of learning about a modeled system's performance characteristics.

Higher-level strategies may use lower-level strategies to implement their specific task.

Throughput: Amount of workload elements being created (or generated), inserted, processed and finished within a specific amount of time. As time is relative (denoted in units, not in seconds or hours), throughput is also relative to the workload insertion rate.

Users are typically interested in the mean throughput over either the whole simulated time or over a specific observation interval.

Turnaround Time: Simulated time for workload elements to be created (or generated), inserted, processed and finished. Turnaround time for user requests are often called 'response time'.

Users of evaluation systems are typically interested in the mean turnaround time for a specific observation period or for the whole simulation period.

Validity: Decision function of performance evaluation experiments to allow for a tool to decide, whether simulation results are valid or not. The validity criterion is based on the reached simulated time in comparison to the target simulated time: If the target time has been reached during an experiment, the experiment is valid, else it is invalid and its result should not be considered. This is to make sure that decisions in the design space do not rely on statistically questionable results.

Workload: General term for the work imposed to a system or model or for the set of all workload elements, respectively.

Workload also has relative quantity meaning, i.e. 'more workload' is understood as a higher amount of workload elements typically inserted for the same time period.

Workload Element: Individual workload request that is created for a specific service and has to be processed by the appropriate resource.

Workload Stream: The series of workload elements departing from one originator. We denote a workload stream to be 'small' or 'narrow' if only few workload elements are originated in a specific period, and 'large' or 'broad' if many workload elements are passed further.

HIT Terms

The originators of HIT defined some term sometimes confusing for people who are used to scientific term or other tools' terms.

The following terms are the most used ones:

OCCUPATION refers to the probability for a new load unit, to find a model or a component unoccupied or occupied by at least one previously inserted load unit. Only the state changes between occupied and unoccupied are measured. The MEAN OCCUPATION value is thus the probability for a new load unit to find a component occupied.

The OCCUPATION measurement stream is updated at every state change between unoccupied and occupied.

POPULATION refers to the number of load units currently in the model or component. The MEAN POPULATION value is thus the average number of load units within the observed modeled object over the simulation's observation time.

The POPULATION measurement stram is updated, when a load element leaves the model or component.

SCHEDULE RATE refers to the rate at which a load unit is transferred from the entry area (the waiting or queueing area of the model or component) to the service area, where it is actually processed. The MEAN SCHEDULE RATE refers thus to the average transition rate, at which load units are scheduled for the service area.

THROUGHPUT refers to the number of load units leaving a simulated model. The MEAN THROUGHPUT value is thus the average number of processed, leaving load units per one time unit over the simulation's observation period.

The THROUGHPUT measurement stream is updated, when a load unit leaves the model.

TURNAROUNDTIME refers to the time, each individual load unit spends within the model or component. The MEAN TURNAROUNDTIME thus refers to the average time for all finished load units spent in the model over the simulation's observation period.

The TURNAROUNDTIME measurement stream is updated, when a load unit leaves the model or component. It thus ignores any load unit either leaving after the observation period or 'starving' within the model.

UTILIZATION refers to the speed of a component of type `server` or `prioserver`. It measures the actual speed of the component due to its current load situation; its measurement stream is updated when the speed of a component is changed.

For components with `shared` or `equal` processing speed specification, UTILIZATION corresponds to the actual speed. If the speed specification is `sdshared` or `sdequal`, UTILIZATION is the standard speed multiplied by the appropriate entry of the speeds array. This may be required when modeling for footprint data or for simplified complex systems.

The MEAN UTILIZATION refers to the average speed of the component; it may yield a value > 1.

Bibliography

[ABB+02] James Aries, Subhankar Banerjee, Marc S. Brittan, Eric Dillon, Janusz S. Kowalik, and John P. Lixvar. Capacity and performance analysis of distributed enterprise systems. *Communications of the ACM*, 45(6):100–105, June 2002.

[Arm01] William Y. Arms. *Digital Libraries*. The MIT Press, Cambridge, MA, 2001.

[ASS96] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, MA, 1996.

[AZG99] Niv Ahituv, Moshe Zviran, and Chanan Glezer. Top management toolbox for managing corporate it. *Communications of the ACM*, 42, April 1999.

[BDE+95] Eike Born, Thomas Delica, Werner Ehrl, Lutz Richter, and Reinhard Riedl. Lydia—characterization of workloads for distributed processing—methodology and guide. *ESPRIT III P8144*, June 1995.

[Bei95] Boris Beizer. *Black-Box Testing—Techniques for Functional Testing of Software and Systems*. Wiley, 1995.

[BEK+98] Barry Boehm, Alexander Egyed, Julie Kwan, Dan Port, Archita Shah, and Ray Madachy. Using the winwin spiral model: A case study. *IEEE Computer*, 31, July 1998.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture*. John Wiley & Son Ltd, 1996.

[BMW93] H. Beilner, J. Mäter, and C. Wysocki. The hierarchical evaluation tool HIT. Technical report, University of Dortmund, Lehrstuhl für Informatik, Germany, 1993.

[BN84] Andres D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[Boe88] Barry Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.

[Bor03] Christine L. Borgman. *From Gutenberg to the Global Information Infrastructure*. The MIT Press, Cambridge, MA, 2003.

[Bur01] Roger Burlton. *Business Process Management—Profiting from Process*. Sams, 2001.

[CFSD90] Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall, and James R. Davin. *RFC 1157 – A Simple Network Management Protocol*, 1990.

[CKN+03] Christian Collberg, Shephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*. Association for Computing Machinery (ACM), 2003.

[Con98] P.C. Consul. *Generalized Poisson Distributions: Properties and Applications*. Marcel Dekker, 1998.

[DG93] Jack J. Dongarra and Wolfgang Gentzsch. *Computer Benchmarks*. North-Holland, 1993.

[Dig98] Tom Digre. Business object component architecture. *IEEE Software*, September/October 1998.

[Dol00] Shlomi Dolev. *Self-Stabilization*. The MIT Press, Cambridge, MA, 2000.

[Dun02] Margaret H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.

[Eig01] Rudolf Eigenmann, editor. *Performance Evaluation and Benchmarking with Realistic Applications*. The MIT Press, 2001.

[Fer78] Domenico Ferrari. Computer systems performance evaluation, 1978.

[Fow99] Martin Fowler. *UML Distilled*. Addison Wesley, 1999.

[GGR99] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining very large databases. *IEEE Computer*, 32, August 1999.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[Gin88] Myron Ginsberg. High-speed and large-scale computing: A panoramic view, 1988.

[Gro01] The Object Management Group. *CORBA Event Service Specification (1.1)*, 2001.

[Gro02] The Object Management Group. *The Common Object Request Broker Architecture Specification (3.0.2)*. The Object Management Group, 2002.

[Hel98] John J. Heldt. *Quality Sampling and Reliability: New Uses for the Poisson Distribution*. Saint Lucie Press, 1998.

[HIS93] Hi-slang reference manual. Technical report, University of Dortmund, Lehrstuhl für Informatik, Germany, 1993.

[Hit93] Hitgraphic user's guide. Technical report, University of Dortmund, Lehrstuhl für Informatik, Germany, 1993.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[HMW02] Hajo Hippner, Melanie Merzenich, and Klaus D. Wilde. *Web Mining in Marketing*. Vieweg, 2002.

[Hol95] David Hollingsworth. The workflow reference model. 1995.

[HPG02] John L. Hennessy, David A. Patterson, and David Goldberg. *Computer Architecture*. Morgan Kaufmann, May 2002.

[HS94] Chao-Ju Hou and K.G. Shin. Load sharing with consideration of future task arrivals in heterogeneous distributed real-time systems. *IEEE Transactions on Computers*, 43(9):1076–1090, September 1994.

[HSFGS99] Brian Henderson-Sellers, Douglas Firesmith, Ian Graham, and A.J.H. Simons. Instantiating the process metamodel. *Journal of Object-Oriented Programming*, June 1999.

[JBR98a] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1998.

[JBR98b] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.

[JBR99a] Ivar Jacobson, Grady Booch, and James Rumbaugh. The unified process. *IEEE Software*, May/June 1999.

[JBR99b] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[Kan02] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. Wiley–IEEE Press, 2002.

[KB98] Wojtek Kazaczynski and Grady Booch. Component-based software engineering. *IEEE Software*, September/October 1998.

[Kop04] Karl Kopper. *The Linux Enterprise Cluster*. No Starch Press, September 2004.

[Kru00] Philippe Kruchten. *The Rational Unified Process*. Addison Wesley, 2000.

[Lan04] Derek Lane. *JUnit—The Definitive Guide*. O'Reilly, 2004.

[LD03] Peter C. Lockemann and Klaus R. Dittrich. *Architektur von Datenbanksystemen*. DPunkt Verlag, 2003.

[LL01] Alessandro Lomi and Erik R. Larsen. *Dynamics of Organizations*. The MIT Press, Cambridge, MA, 2001.

[Mey00] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall PTR, 2000.

[Mit98] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA, 1998.

[Mul93] Sape Mullender. *Distributed Systems*. ACM Press, 1993.

[MvNV+99] Jason Maassen, Rob van Nieuwpoort, Ronald Veldema, Henri E. Bal, and Aske Plaat. An efficient implementation of java's remote method invocation. *Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, 5(1):173–182, 1999.

[oGC00] Office of Government Commerce. *ITIL Service Support*. The Stationary Office Books, 2000.

[oGC01] Office of Government Commerce. *ITIL Service Delivery*. The Stationary Office Books, 2001.

[oGC02] Office of Government Commerce. *ITIL Application Management*. The Stationary Office Books, 2002.

[OH98] R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, 1998.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Cambridge, MA, 2002.

[PW93] Udo W. Pooch and James A. Wall. *Discrete Event Simulation: A Practical Approach*. CRC Press, Computer Engineering Series, Boca Raton FL, USA, 1993.

[Rie99] Reinhard Riedl. The impact of workload on simulation results for distributed transaction processing. In Peter Sloot, Martin Bubak, Alfons Hoekstra, and Bob Hertzberger, editors, *Proceedings of the 7th International Conference on High-Performance Computing and Networking (HPCN)*, volume 7. Springer, Berlin, 1999.

[Rie01] Reinhard Riedl. Need for trace benchmarks. In Rudolf Eigenmann, editor, *Performance Evaluation and Benchmarking with Realistic Applications*, pages 220–256. The MIT Press, 2001.

[Ros96] Marshall T. Rose. *The Simple Book: An Introduction to Internet Management*. Prentice Hall, 1996.

[Sal96] R. Salz. *OSF RRC 63.3, DCE 1.2 Content Overview*, 1996.

[Som92] Ian Sommerville. *Software Engineering*. Addison Wesley, 1992.

[Ste97] W. Richard Stevens. *RFC 2001 – TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, 1997.

[Ste01] Thomas J. Sterling. *Beowulf Cluster Computing with Linux*. The MIT Press, Cambridge, MA, 2001.

[Sun97] *White Paper—Java Remote Method Invocation—Distributed Computing for Java*, 1997.

[Tan01] Andrew Tanenbaum. *Modern Operating Systems, 2nd edition*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2001.

[vdAvH02] Wil van der Aalst and Kees van Hee. *Workflow Management*. The MIT Press, Cambridge, MA, 2002.

[Wey98] Elaine J. Weyuker. Testing component-based software: A cautonary tale. *IEEE Software*, September/October 1998.

[Wil88] Robert B. Wilhelmson. *High-Speed Computing: Scientific Applications and Algorithm Design*. University of Illinois, Urbana-Champaign, 1988.

[ZSN01] Mazen Zari, Hossein Saiedian, and Muhammad Naeem. Understanding and reducing web delays. *IEEE Computer*, 34, December 2001.

Peter Lukas Weibel was born in 1969 in Liestal, Switzerland. From 1989 to 1995 he studied Computer Science and Economics at the University of Zurich, Switzerland.

From 1995 to 1999 he was with *Systor AG* in Basel, Switzerland, first as research engineer, then as software engineer specialized in middleware and distributed systems architecture. In 1999 he joined *itheca Consulting Ltd.*, a consulting company in Basel and Zurich, Switzerland, where he worked as senior information systems consultant and management consultant for clients in the financial industry and in the hardware industry.

From 1997 on, Peter Weibel was guest member of the *Software Architecture and Systems Group* of the University of Zurich with Prof. Dr. L. Richter and Dr. R. Riedl. This dissertation thesis is a result of his research activities in this group.